

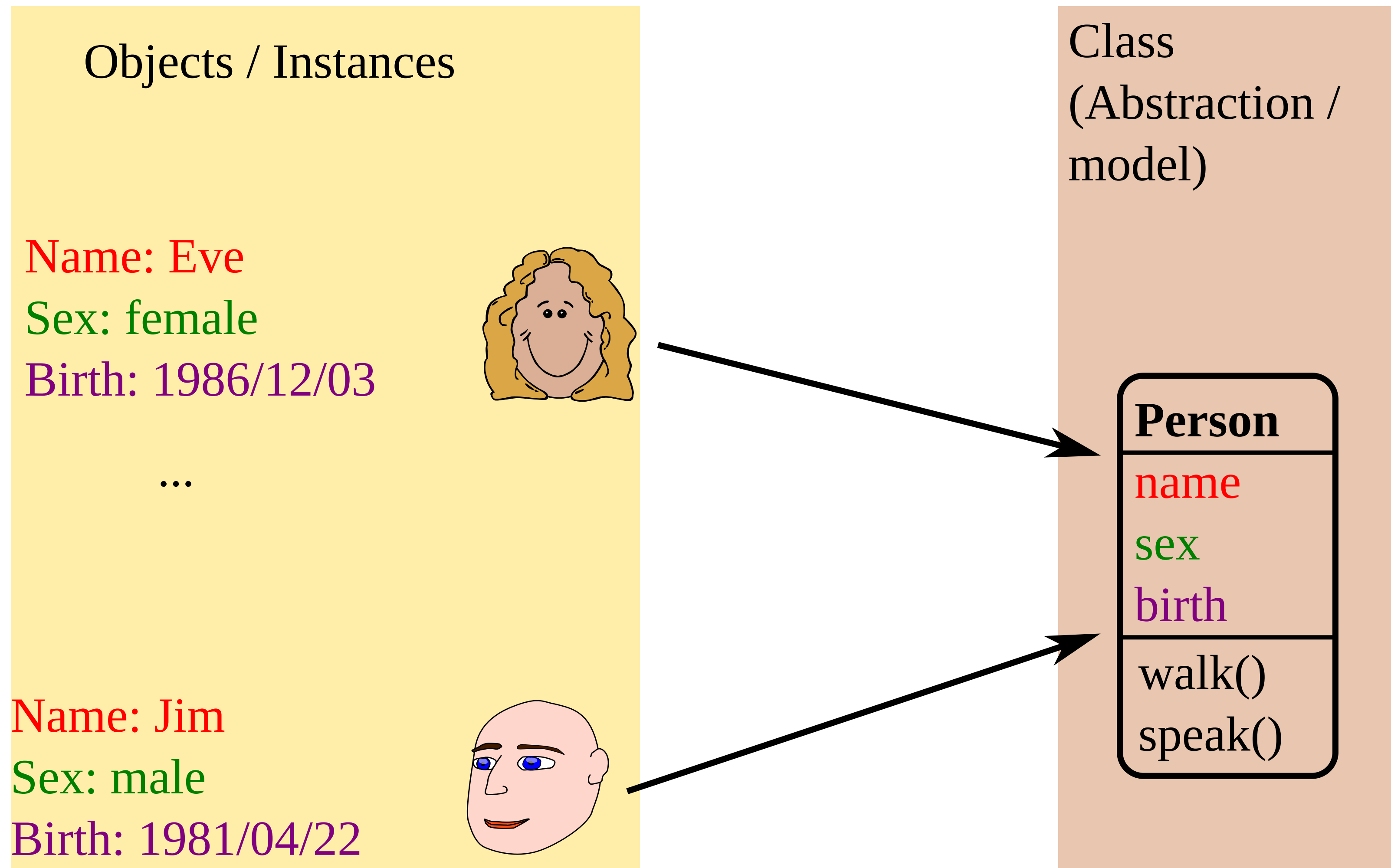
Objects and Classes

The concept of classes: Blueprints for objects.

Example: Rectangles

UML diagram visualization.

Instances of a Class



General class structure

Person	The classes name
name sex birth	Attributes
walk() speak()	Methods

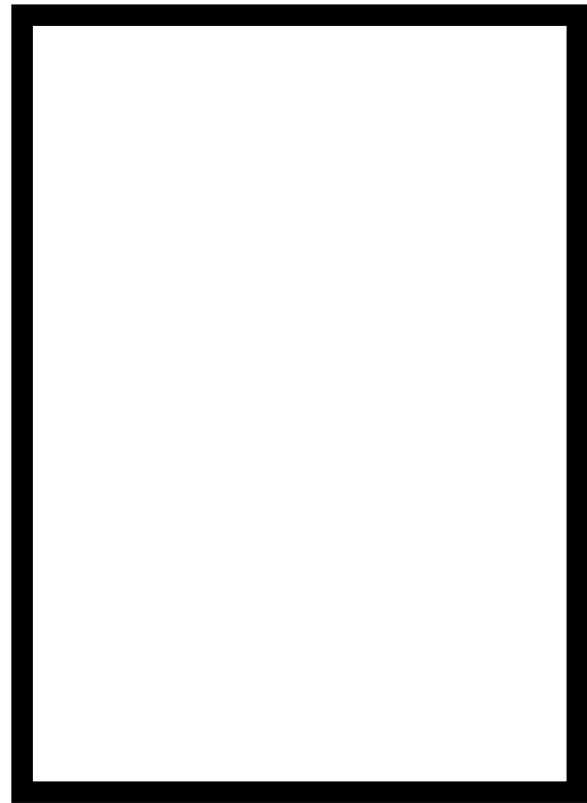
What's a class anyway?

In object oriented languages **classes**:

- are blueprints for objects.
- contain attributes and methods.
- allow for implementation hiding.
- allow for tailored access to methods and attributes.

Rectangle objects

width=20



height=28

solid border

width=28

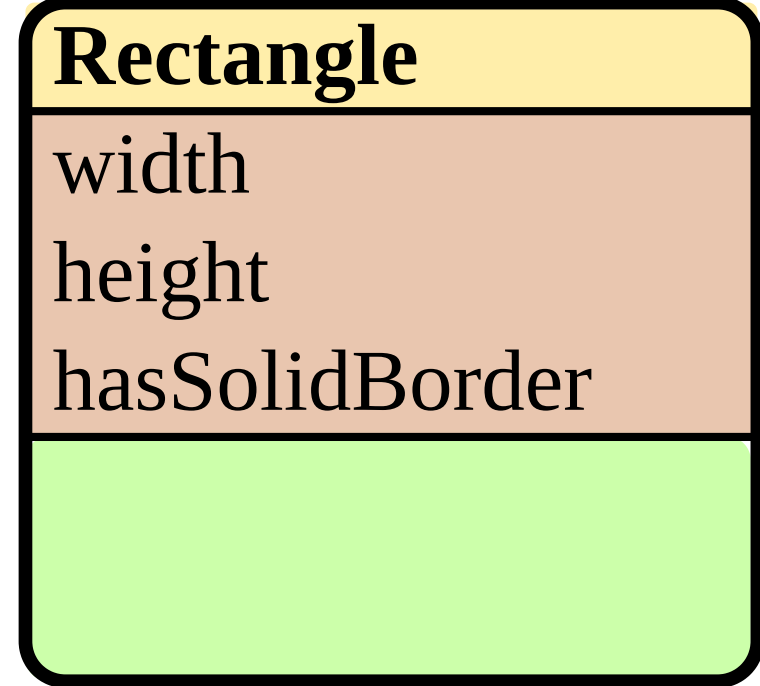


height=10

dashed border

A class describing rectangles

```
public class Rectangle {  
    int width;  
    int height;  
  
    // solid or dashed:  
    boolean hasSolidBorder;  
}
```



A UML class diagram for the Rectangle class. The diagram is a rounded rectangle divided into three horizontal compartments. The top compartment has a yellow background and contains the class name "Rectangle" in bold black text. The middle compartment has a light orange background and contains the attributes "width", "height", and "hasSolidBorder" listed vertically. The bottom compartment has a light green background and is currently empty.

Rectangle
width
height
hasSolidBorder

Rectangle class and instances

Class

```
public class Rectangle {  
    int width;  
    int height;  
    boolean hasSolidBorder;  
}
```

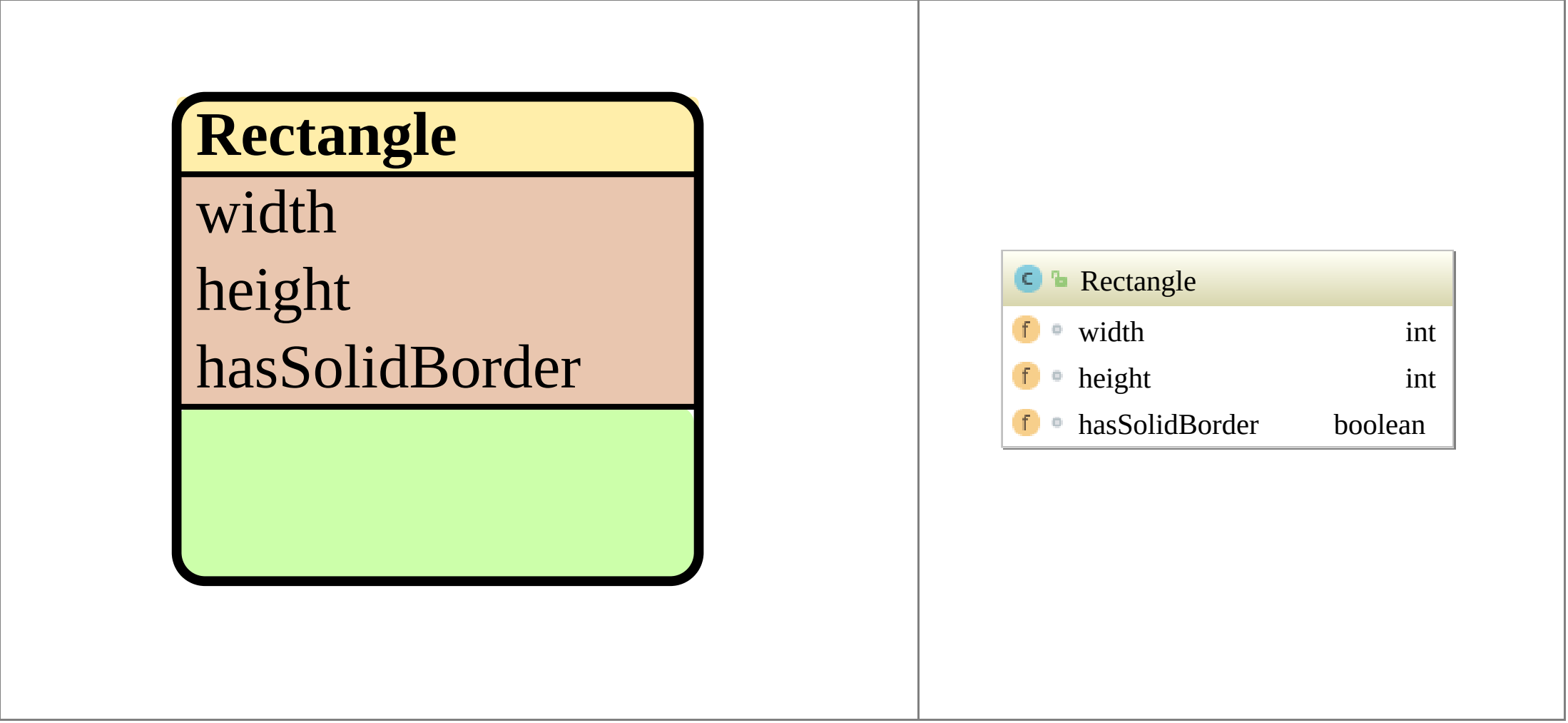
is instance of

**Objects
(Instances)**

35
false 15

28
true 40

Generated diagrams



Objects and Classes

- ⇒ Working with objects

Classes and creation of related instances.

The «new» operator.

Assignment of attribute values. Working with «null».

The **new** operator: Creating rectangle instances

```
Rectangle dashedRectangle = new Rectangle();
```

```
Rectangle solidRectangle = new Rectangle();
```

```
...
```

Syntax creating instances

```
new class-name ([argument 1[, argument 2] ...] )
```

Wording examples:

- “Create an instance of class `Rectangle`”.
- “Create a `Rectangle` object.”
- “Create a `Rectangle`”.

Assigning attribute values to class instances

```
Rectangle dashedRectangle = new Rectangle();  
  
dashedRectangle.width = 28;  
dashedRectangle.height = 10;  
dashedRectangle.hasSolidBorder = false;
```

Syntax accessing object attributes:

```
variable.attributeName = value;
```

Instance memory representation

```
Rectangle r =  
    new Rectangle();  
r.width = 20;  
r.height = 30;  
r.hasSolidBorder = true;
```

...							
0	0	0	1	0	1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1
...							

References and null

```
Rectangle r = new Rectangle();// Creating an object  
r.width = 28; // o.K.  
  
r = null;// removing reference to object  
  
r.width = 28; // Runtime error: NullPointerException (/
```

```
Exception in thread "main" java.lang.NullPointerException  
    at de.hdm_stuttgart.mi.classdemo.App.main(App.java:17)
```

Checking for object presence

```
Rectangle r;
```

```
... // possible object assignment to variable r.
```

```
if (null == r) {  
    System.out.println("No rectangle on offer");  
} else {  
    System.out.println("Width:" + r.width);  
}
```

Objects and Classes

⇒ Packages

Significance for larger projects.

Fully qualified vs. «`import`».

`java.lang` and other exceptions.

Why packages ?

- Grouping of related classes (e.g. subsystems).
- Structuring big systems.
- Provide access restrictions using:
`public, private and protected` modifier
- Resolving class name clashes. Example:
`java.lang.String` vs. `my.personal.String`

Rules and conventions

- Package names below **java.** are reserved.
- Package names should not start with **javax.** either.
- Package names must not contain operators:
`mi.hdm-stuttgart.de --> de.hdm_stuttgart.mi.`
- Packages should start with reversed **DNS** avoiding clashes.

Fully qualified class name vs. **import**

Fully qualified class name:

```
java.util.Scanner ❶ scanner =          // Clumsy and  
    new java.util.Scanner ❷(System.in); // redundant
```

Using **import**:

```
import java.util.Scanner; ❶  
  
public class Q {  
    public static void main(String[] args) {  
        Scanner ❷ scanner = new Scanner ❷(System.in);  
        ...  
    }  
}
```

Don't be too lazy!

Bad

```
import java.util.*;

public class Q {
    public static void
        main(String[] args) {
        Scanner s =
            new Scanner(System.in);
        Date today = new Date();
    }
}
```

Good

```
import java.util.Scanner;
import java.util.Date;

public class Q {
    public static void
        main(String[] args) {
        Scanner s =
            new Scanner(System.in);
        Date today = new Date();
    }
}
```

Special: Classes in package java.lang

```
import java.lang.String; ❶ // Optional
import java.util.Scanner; ❷ // Required
public class Q {

    public static void main(String[] args) {
        String message = "Hello!";
        Scanner ❸ s = new Scanner(System.in);
    }
}
```

Class, package and file system

The screenshot shows an IDE interface. The top breadcrumb navigation shows the path: `my` > `first` > `javapackage` > `Print`. Below this is a toolbar with icons for binary, search, run, debug, and other IDE functions. On the left is a 'Project' view showing a tree structure: `target` (yellow), `classes` (grey), `my` (yellow), `first` (yellow), `javapackage` (yellow), and `Print.class` (blue with a lock icon). A black circle with the number '2' is next to `Print.class`. The main editor window shows a tab for `Print.java`. The code in the editor is as follows:

```
1 package my.first.javapackage;
2
3 public class Print {
4     public void print(int i) {
5         System.out.println("ir");
6     }
7 }
```

A black circle with the number '1' is next to the `package` statement on line 1. The code is color-coded: `package` is blue, `my.first.javapackage` is black, `public` is blue, `class` is blue, `Print` is black, `void` is blue, `print` is black, `int` is blue, `System.out` is purple, and `println` is green.

Source hierarchy view

Recent

Home

Desktop

Trash

klausur

Documents

Music

GoikLectures

Name	Size	Type
1 java	2 items	Folder
2 my	1 item	Folder
3 first	1 item	Folder
4 javapackage	1 item	Folder
5 Print.java	413 bytes	Text File

```
package my.first.javapackage
    public class Print {
        ...
    }
```

Objects and Classes

⇒ Object methods

Object behaviour, state transitions, derived values.

Selected examples both of state-changing and state-conserving methods.

Rectangle sample methods.

Object methods

Change an object's state.

Example: Scale a rectangle.

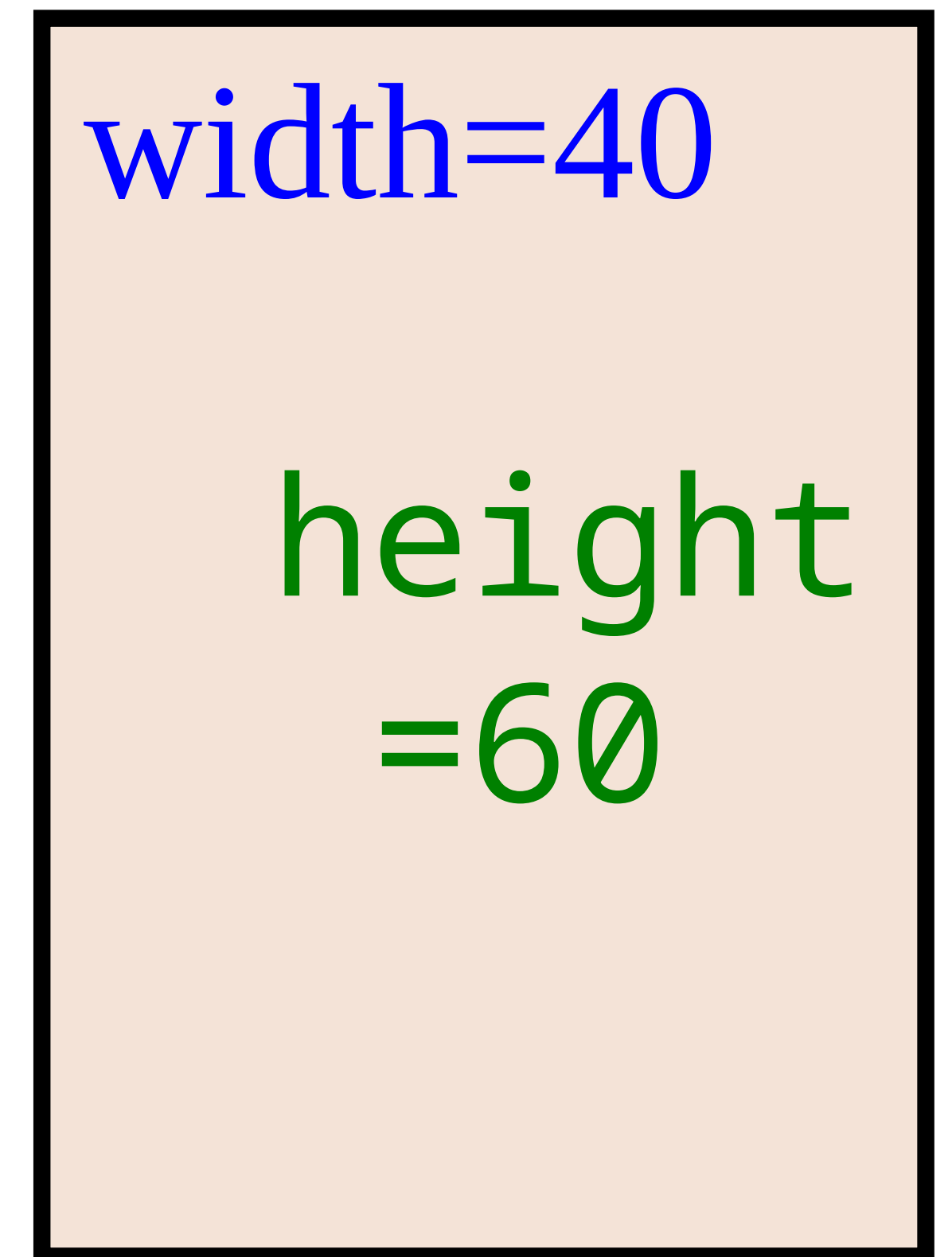
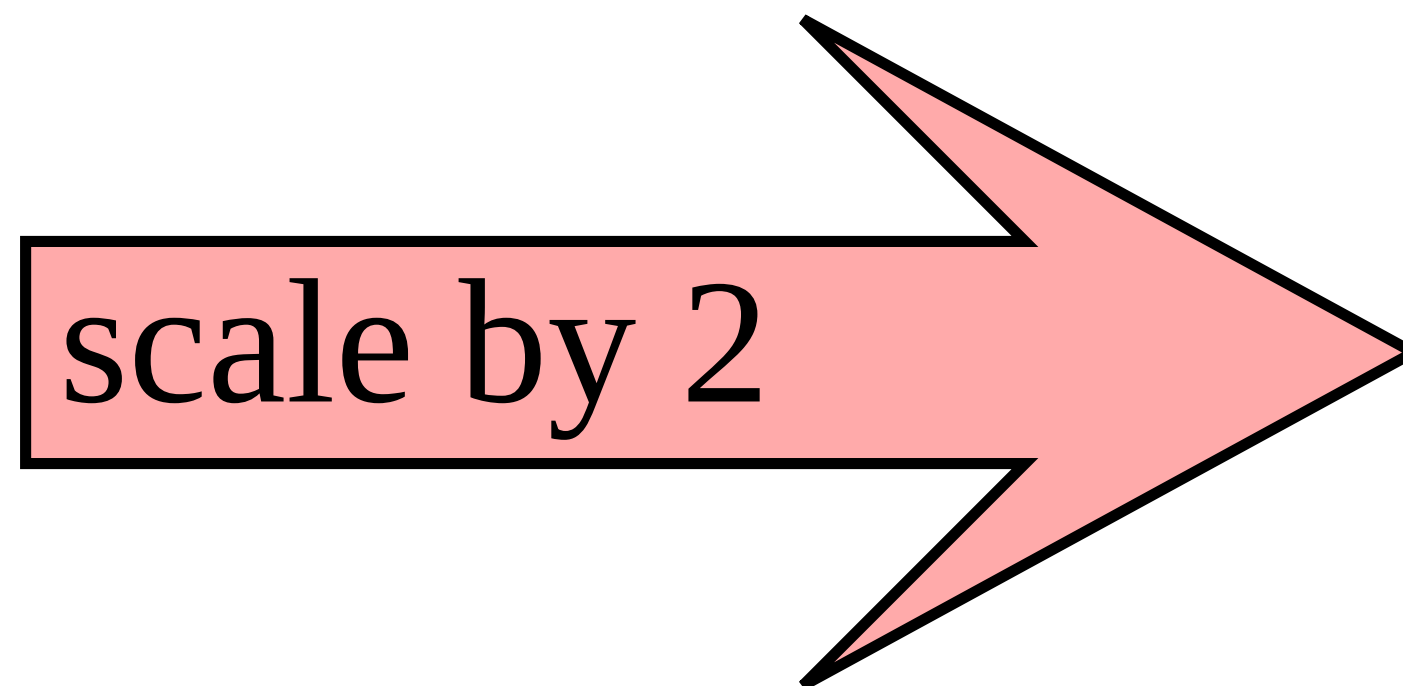
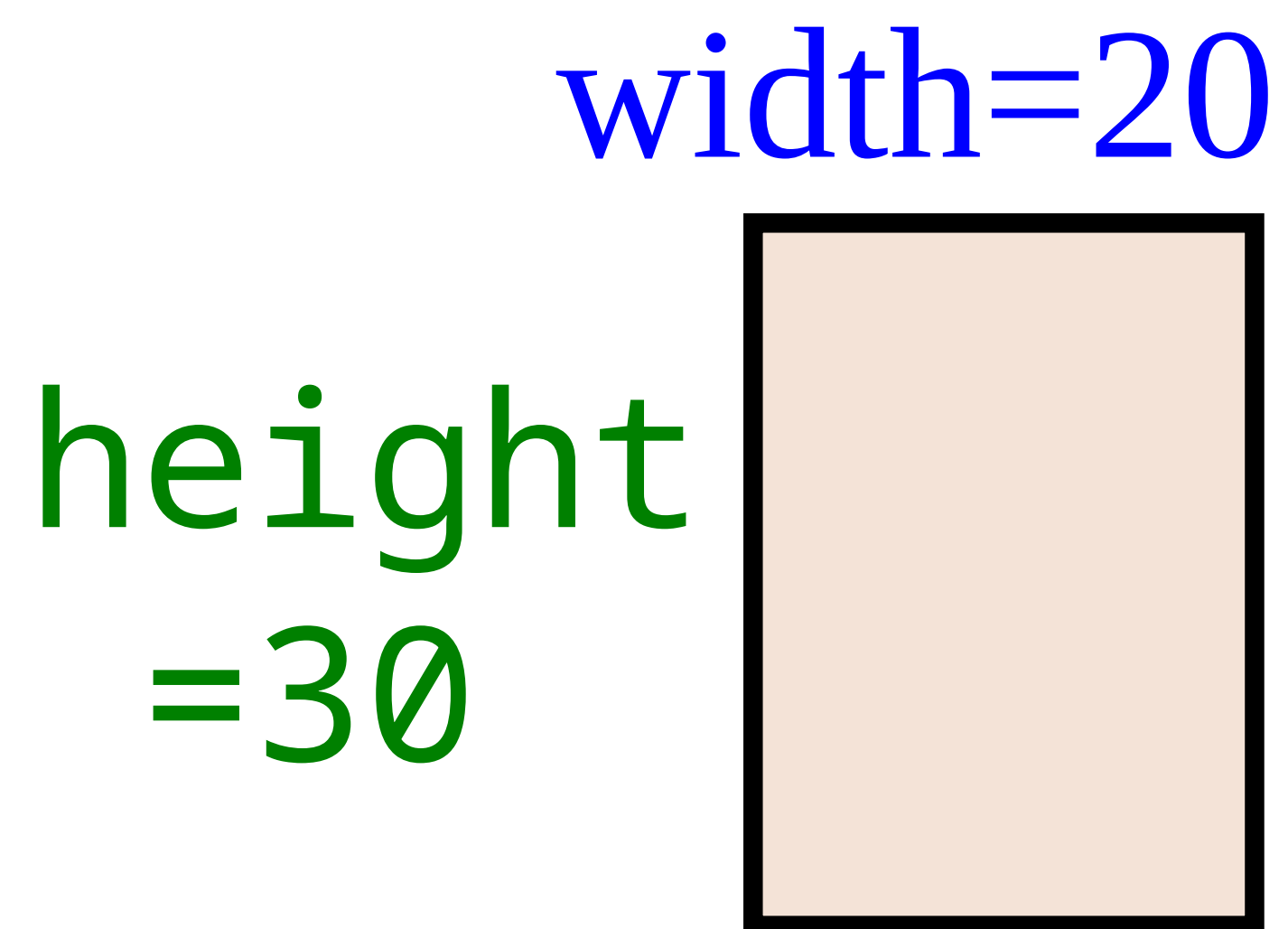
Get dependent values

Example: Calculate a rectangle's perimeter.

Combined

Scale a rectangle and calculate its new perimeter.

Scaling a rectangle



Scaling method implementation

```
public class Rectangle {  
    int width, height;  
    boolean hasSolidBorder;  
  
    public void scale (int factor)  
        width *= factor;  
        height *= factor;  
    }  
}
```

Rectangle		
f	width	int
f	height	int
f	hasSolidBorder	boolean
m	scale(int)	void

Scaling method signature

```
void ❶ scale (int factor ❷) {  
...}
```

- ❶ No value is being returned to caller.
- ❷ A single value of type `int` is being provided as method argument.

Using the `scale( . . . )` method

```
Rectangle r = new Rectangle();  
r.width = 33;  
r.height = 22;  
  
r.scale(2);  
  
System.out.println("width=" + r.width);  
System.out.println("height=" + r.height);
```

```
width=66  
height=44
```

Method definition syntax

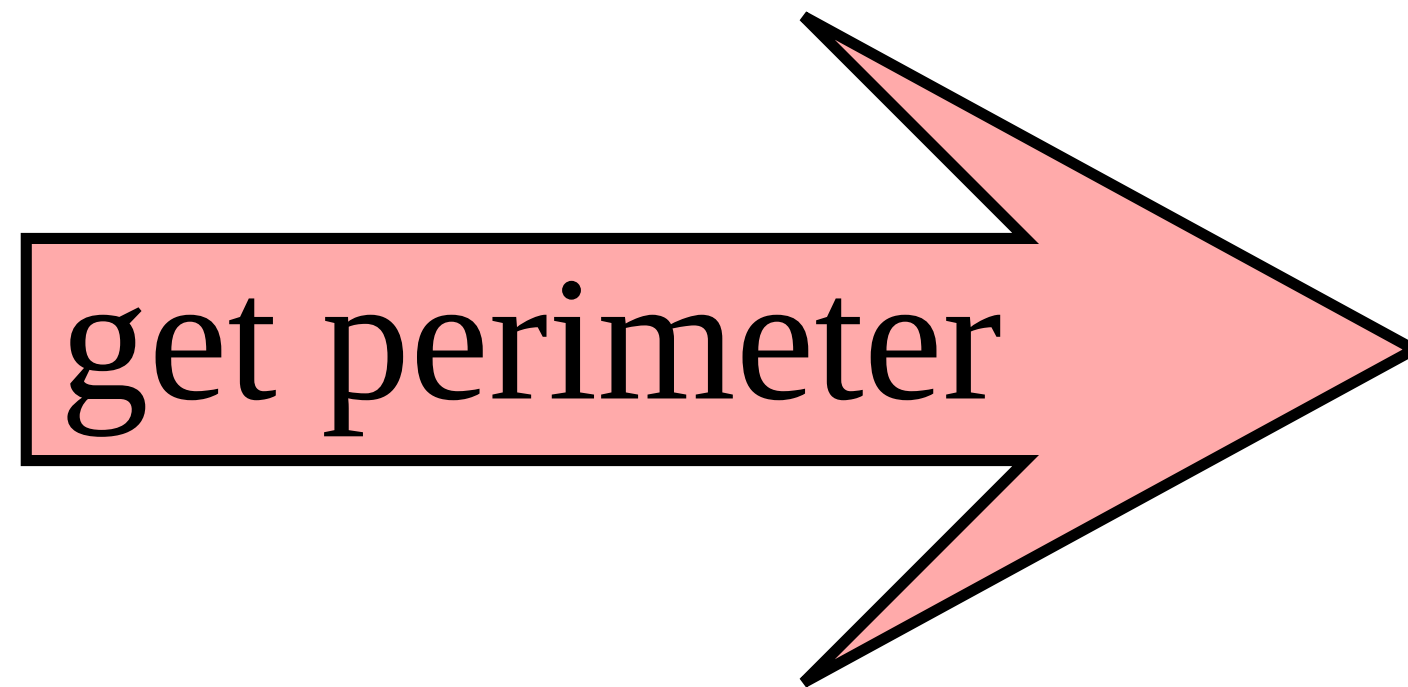
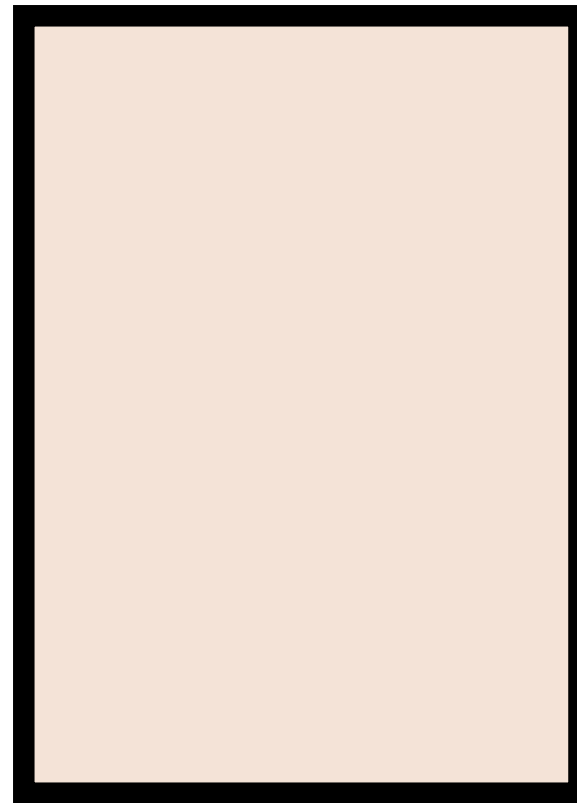
```
public ❶ void ❷ scale❸ (int factor ❹) { ❺  
    width *= factor; ❻  
    height *= factor;  
}
```

```
[access modifier] ❶ return_type ❷ methodName ❸ ([arguments] ❹) { ❺  
    [statement(s)] ❻  
}
```

A rectangle's perimeter

width=20

height
=30



100

getPerimeter() method implementation

```
public class Rectangle {  
    int width, height;  
    boolean hasSolidBorder;  
  
    public void scale (int factor) { ...  
  
    public int getPerimeter() {  
        return 2 * (width + height);  
    }  
}
```

		Rectangle	
		width	int
		height	int
		hasSolidBorder	boolean
		scale(int)	void
		getPerimeter()	int

Using Rectangle.getPerimeter()

```
Rectangle r = new Rectangle();  
  
r.width = 33;  
r.height = 22;  
  
System.out.println("Perimeter=" + r.getPerimeter());
```

Perimeter=110

Followup exercises

90. Compile time error

91. An Address class

Objects and Classes

- ⇒ Object methods

- ⇒ Encapsulation and access control

- Justifying access restrictions.

- `public`, `protected` «package local» and `private`.

- Best practices.

Access control: Overall objectives

- Fine-grained control on attributes and methods.
- Support **encapsulation / Information hiding**.

Purpose: Hide implementation details within class or package

Example: Two ways implementing a day's time

```
public class DayTime {  
    private int ① minutes_since_0_00;  
  
    public int getMinute() { ②  
        return minutes_since_0_00 % 60;  
    }  
    public int getHour() { ②  
        return minutes_since_0_00 / 60;  
    }  
}
```

```
public class DayTime {  
    private int minute, hour; ①  
  
    public int getMinute() { ②  
        return minute;  
    }  
    public int getHour() { ②  
        return hour;  
    }  
}
```

Access violation

No access to private field of alien class Time:

```
public class Q {  
    public static void main(String[] args) {  
  
        Time t = new Time();  
  
        // Error: 'minutes_since_0_00' has private access in 'Time'  
        t.minutes_since_0_00 = 371;  
    }  
}
```

Access rules

Access Level	Other package	Child class	Same package	Same class
public	yes	yes	yes	yes
protected	no	yes	yes	yes
Default	no	no	yes	yes
private	no	no	no	yes

“Tips on Choosing an Access Level”

- Use the most restrictive access level that makes sense for a particular member.
- Use `private` unless you have a good reason not to.
- Avoid `public` fields except for constants. Public fields tend linking to a particular implementation and limit your flexibility in changing your code.

Followup exercises

- 92. Understanding access control
- 93. Explaining times

Overview

Objects and Classes

- ⇒ Object methods

- ⇒ Getter and setter methods

Direct access vs. setter method

Direct access	Setter access
<pre>public class Time { public int hour, minute; }</pre> <pre>Time time = new Time(); time.hour = 17; time.minute = 45;</pre>	<pre>public class Time { private int hour, minute; public void setTime(int h, int m) { minute = m; hour = h; } }</pre> <pre>Time time = new Time(); time.setTime(17, 45);</pre>

Why adding setter methods?

- Allow for change of implementation.
- Allow for non-business logic concerns.

Examples:

- Logging
- Adding persistence (Databases)

```
public class Time {  
    private int hour, minute;  
    public void setTime(int h, int m) {  
        minute = m;  
        hour = h;  
        System.out.println("Time has been set to "  
            + hour + ":" + minute);  
    }  
}
```

Implementation change: Minutes only, no hours

Direct access	Setter / getter access
<pre data-bbox="537 804 1187 1083">public class Time { // Minutes since 00:00 public int minute; }</pre> <pre data-bbox="537 1125 1187 1234">final Time time = new Time(); time.minute = 17 * 60 + 45;</pre>	<pre data-bbox="1338 804 2401 1272">public class Time { private int minute; // Minutes since 00. public void set(int h, int m) { minute = m + 60 * h; } public int getMinute(){/* coming soon */} public int getHour(){/* coming soon */} }</pre> <pre data-bbox="1338 1314 2401 1423">final Time time = new Time() time.setTime(17, 45);</pre>

94. Implementing getter methods

Objects and Classes

- ⇒ Object methods

- ⇒ Signatures

Type signatures.

Method signatures: Prerequisite understanding method overloading.

Defining type signatures

```
boolean 1 startsWith(String prefix, int toffset) 2
```

¹ Return type boolean

² Arguments among with their respective types and order:

1. Type String

2. Type int

Type signature examples

Method	Return type	Argument type list
<code>void print()</code>	<code>void</code>	<code>(void)</code>
<code>int add (int a, int b)</code>	<code>int</code>	<code>(int, int)</code>
<code>int max (int a, int b)</code>	<code>int</code>	<code>(int, int)</code>
<code>void print (int a, float b)</code>	<code>void</code>	<code>(int, float)</code>
<code>void display (float a, int b)</code>	<code>void</code>	<code>(float, int)</code>

Defining method signatures

```
boolean startsWith❶(String prefix, int toffset)❷
```

- ❶ Method name `startsWith`.
- ❷ Number of arguments among with their respective types and order.

Method signature examples

Method	Method name	Method signature
<code>void print()</code>	print	(void)
<code>int add (int a, int b)</code>	add	(int, int)
<code>int max (int a, int b)</code>	max	(int, int)
<code>void print (int a, float b)</code>	print	(int, float)
<code>void display(float a, int b)</code>	display	(float, int)

95. Method signature variants

Objects and Classes

- ⇒ Object methods

- ⇒ Method overloading

- Same name, different argument types.

- «C» programming language comparison.

- Common examples.

Method overloading: Same name, different signature

```
public class Print {  
    public void print() {           // (void)  
        System.out.println("No argument");  
    }  
    public void print(int i) {      // (int)  
        System.out.println("int value " + i);  
    }  
    public void print(double d) {   // (double)  
        System.out.println("double value " + d);  
    }  
    public void print(int i, int j) { // (int, int)  
        System.out.println("Two int values " +  
            i + " and " + j);  
    }  
}
```

```
Print p = new Print();  
p.print();  
p.print(33);  
p.print(4.333);  
p.print(-1, 7);
```

```
No argument  
int value 33  
double value 4.333  
Two int values -1 and 7
```

Overloading, alternate names

- Static polymorphism.
- Compile time binding.
- Early binding.

No overloading in "C"

```
#include <stdio.h>

void print() {
    printf("No argument\n");
}

void print(int i) {                /* Error: redefinition of 'print' */
    printf("int value %d\n", i);
}

void main(void) {
    print();
    print(33);
}
```

”C“ requires unique function names

Code in file <code>print.c</code>	Compile / execute
<pre>#include <stdio.h> void print() { printf("No argument\n"); } /* Different function name */ void printIntValue(int i) { printf("int value %d\n", i) } void main(void) { print(); printIntValue(33); }</pre>	Compiling <code>print.c</code> to executable file <code>print</code> <pre>> cc -o print print.c</pre>
	Executing file <code>print</code> <pre>> ./print No argument int value 33</pre>

No distinction on return type

```
public class Person {  
    String getDetails() { return "dummy";}  
    int getDetails() { return 1;} // Error: 'getDetails()' is already  
                                // defined in 'Person'  
}
```

Return type	Method signature	
	Method name	Argument type list
String	getDetails	(void)
int	getDetails	(void)

Only method signature support in **Java™** ignoring return type.

Method signatures rationale

In **Java™** method signatures allow for uniquely addressing a method within a given class e.g.:

The method named `print` having an `int` argument followed by a `double`:

```
print(int, double)
```

Method signatures rationale

```
Print p = new Print();
```

```
p.print(3.14);
```

```
public class Print {
```

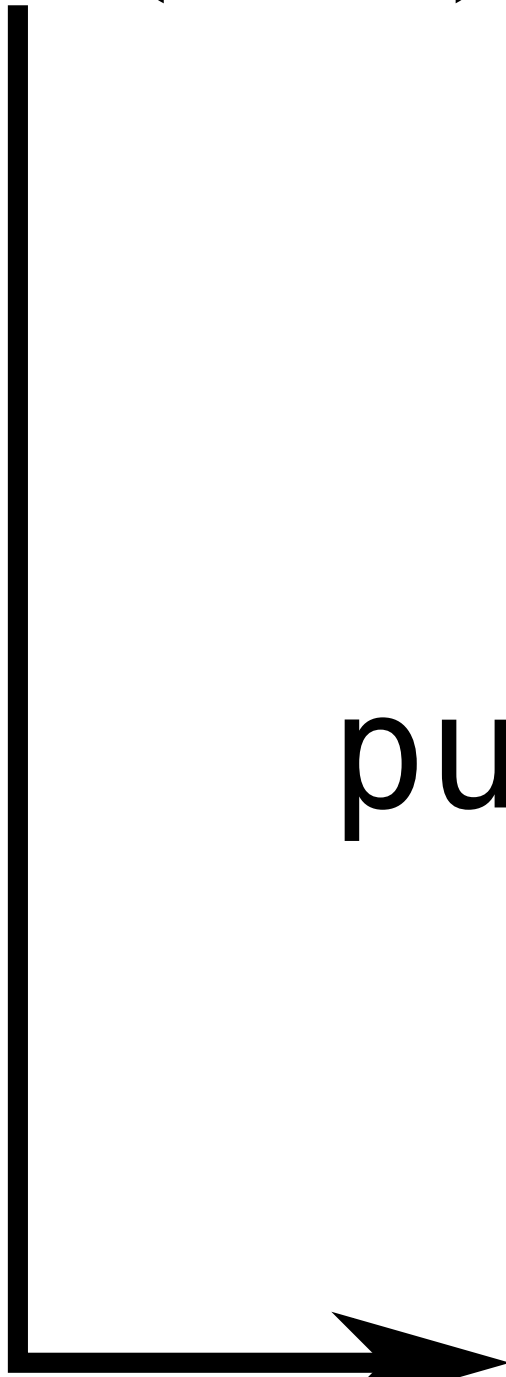
```
    public void print(int i) {...}
```

```
    public void print() {...}
```

```
    public void print(double d) {...}
```

```
    public void print(int i, int j) {...}
```

```
}
```

A diagram consisting of a vertical line extending downwards from the argument '3.14' in the call 'p.print(3.14);', followed by a horizontal line that ends in an arrow pointing to the 'print(double d)' method signature in the class definition.

96. Will a match be found?

Example: System.out.print(...)

print(boolean b)	print(char c)
------------------	---------------

print(char[] s)	print(double d)
-----------------	-----------------

print(float f)	print(int i)
----------------	--------------

print(long l)	print(Object obj)
---------------	-------------------

print(String s)	
-----------------	--

Objects and Classes

- ⇒ Object methods

- ⇒ Constructors

- Combined instance creation and initialization.

- Constructors are methods, but ...

- Constructor overloading and a class' default constructor.

Creating and initializing rectangles

```
int a;  
a = 33;
```

```
Rectangle r = new Rectangle();  
  
r.width = 28;  
r.height = 10;  
r.hasSolidBorder = false;
```

Combining statements desired:

```
int a = 33; // works!
```

```
Rectangle r = new Rectangle(28, 10, false); // how ???
```

Defining a constructor

```
public class Rectangle {  
    int width, height;  
    boolean hasSolidBorder;  
    ...  
    public ❶ Rectangle ❷ (int width, int height, boolean hasSolidBorder){  
        this.width = width;  
        this.height = height;  
        this.hasSolidBorder = hasSolidBorder;  
    }  
}
```

Constructor syntax

```
[access modifier] constructorName (listOfArguments) {  
    [constructor body]  
}
```

Empty argument list

Default constructor e.g. `obj = new MyClass()`.

Non-empty argument list

Non-default constructor e.g. :

```
obj = new String("xyz");
```

Constructors

- Can only be executed on object creation.
- Are being called prior to any non-constructor method.
- Only one of potentially multiple constructors will be executed exactly one time.

However **nesting** is possible.

Multiple overloaded constructors

```
public class Rectangle {  
    int width, height;  
  
    public Rectangle() {  
        width = height = 1;  
    }  
    public Rectangle(int width, int height) {  
        this.width = width;  
        this.height = height;  
    }  
    public Rectangle(int widthAndHeight) {  
        width = height = widthAndHeight;  
    }  
}
```

  Rectangle	
  width	int
  height	int
  Rectangle()	
  Rectangle(int, int)	
  Rectangle(int)	

Constructor calls within constructor

```
public class Rectangle {  
    int width, height;  
  
    public Rectangle(int width  
                    int height){  
        this.width = width;  
        this.height = height;  
    }  
    public Rectangle() {  
        width = height = 1;  
    }  
    public Rectangle(  
        int widthAndHeight) {  
        width = height =  
            widthAndHeight;  
    }  
}
```

```
public class Rectangle {  
    int width, height;  
  
    ▶ public Rectangle(int width,  
                      int height){  
        this.width = width;  
        this.height = height;  
    }  
    public Rectangle() {  
        ◀— this(1, 1); ①  
    }  
    public Rectangle(  
        int widthAndHeight) {  
        ◀— this(widthAndHeight,  
                widthAndHeight); ②  
    }  
}
```

Instances by overloaded constructors

```
Rectangle standard = new Rectangle(); // 1 x 1  
Rectangle square = new Rectangle(2); // 2 x 2  
Rectangle individual = new Rectangle(2, 7); // 2
```

		Rectangle
		width
		int
		height
		int
		Rectangle()
		Rectangle(int, int)
		Rectangle(int)

No constructor vs. default constructor

Equivalent: `Rectangle r = new Rectangle();`

```
public class Rectangle {  
    int width, height;  
    boolean hasSolidBorder;  
  
    // Default constructor, empty body  
    public Rectangle ( ){}  
}
```

```
public class Rectangle {  
    int width, height;  
    boolean hasSolidBorder;  
  
    // No constructor at all  
}
```

Non - default constructor, but no default constructor

```
public class Rectangle {  
    int width, height;  
    boolean hasSolidBorder;  
  
    // Non-default constructor  
    public Rectangle(int width,  
                    int height,  
                    boolean hasSolidBorder){  
        this.width = width;  
        this.height = height;  
        this.hasSolidBorder =  
            hasSolidBorder;  
    }  
  
    // No defined default constructor  
}
```

```
// o.K.: Using non-default  
// constructor.
```

```
Rectangle r =  
    new Rectangle(3, 6, false)
```

```
// Wrong: Default constructor  
// undefined, but non-default  
// constructor present.
```

```
Rectangle r = new Rectangle()
```

Followup exercises

- 97. Modeling geometry objects: Rectangles
- 98. Modeling circles
- 99. Adding translations and SVG export.
- 100. Extending the employee example.
- 101. Refining access to an employee's attributes
- 102. File system representation
- 103. Your personal `String` class

Objects and Classes

- ⇒ Object methods

- ⇒ Scopes

- Class, method and block scopes.

- Documentation, «good» variable names and the `this` keyword.

Circle and variable scopes

```
public class Circle {  
    private double radius;  
  
    public Circle(double r){  
        radius = r;  
    }  
  
    public double getDiameter() {  
        return 2 * radius;  
    }  
}
```

Scope of constructor argument variable »**r**« is limited to constructor

Documenting classes and methods

```
/** Representing circles.  
*/  
public class Circle {  
    private double radius;  
  
    /** Creating a circle.  
    * @param r representing the circle's radius  
    */  
    public Circle(double r) {  
        radius = r;  
    }  
    public double getDiameter() {  
        return 2 * radius;  
    }  
}
```

Constructor Detail

Circle

**Non-self-explanatory
variable name »r«**

```
public Circle(double r)
```

Creating a circle.

Parameters:

r - representing the circle's radius

Constructor Detail

Circle

**Self-explanatory
variable name »radius«**

```
public Circle(double radius)
```

Creating a circle.

Parameters:

radius - Circle's size

Refactoring «r» ⇒ «radius»

```
/** Representing circles. */
public class Circle {
    private double radius;

    /** Creating a circle.
     * @param radius Circle's size
     */
    public Circle(double radius) {
        radius = radius;
    }
    public double getDiameter() {
        return 2 * radius;
    }
}
```

Constructor Detail

Circle

public Circle(double radius)

Creating a circle.

Parameters:

radius - Circle's size

Scope assignment problem

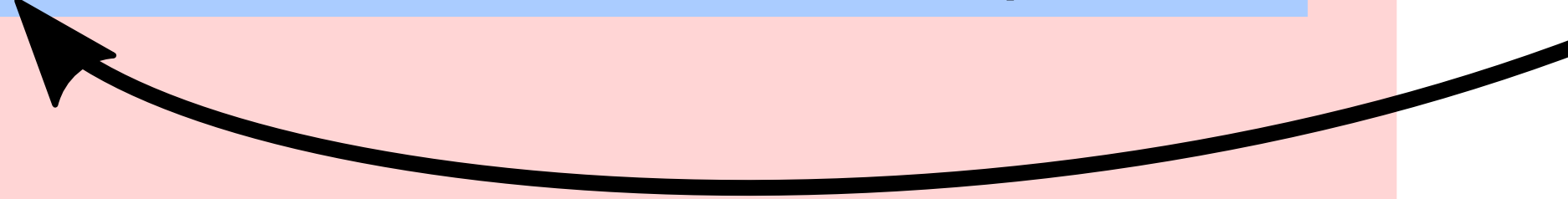
```
/** Representing circles.
 */
public class Circle {
    private double radius;

    /** Creating a circle.
     * @param radius Circle's size
     */
    public Circle(double radius) {
        radius = radius; // Warning: Variable 'radius' is assigned to itself.
    }
    public double getDiameter() {
        return 2 * radius;
    }
}
```

this overriding method scope

```
public class Circle {  
    private double radius;  
  
    public Circle(double radius) {  
        this.radius = radius;  
    }  
  
    public double getDiameter() {  
        return 2 * radius;  
    }  
}
```

The »**this**« keyword relates to **instance scope** resolving method scoped **radius** from instance scoped **radius**.



104. Constructors variable names and “this”.

Objects and Classes

- ⇒ Class members and methods

Motivating class attributes and methods.

Example: A club and its affiliated members.

Syntax of `static` and employment considerations

Why do we require an instance?

```
public class Helper {  
    // Find the larger of two values  
    public int maximum(int a, int b)  
        if (a < b) {  
            return b;  
        } else {  
            return a;  
        }  
}
```

```
final Helper instance = new Helper();  
  
// Why do we need an instance just for  
// computing the maximum of two values?  
System.out.println("Maximum: " +  
    instance.maximum(-3, 5));
```

Maximum: 5

Observation: The instance's state is irrelevant for finding the maximum of two values.

Solution: Replace instance method by class method using **static**

```
public class Helper { ❶  
    static ❷ public int maximum(int a, int b)  
    {  
        if (a < b) {  
            return b;  
        } else {  
            return a;  
        }  
    }  
}
```

```
// No instance required any longer  
System.out.println("Maximum: " +  
    Helper.maximum(-3, 5)); ❶
```

Maximum: 5

Club membership objectives

- Each club member has got a name.
- Each member has got an ascending unique membership number.
- The overall club's member count needs to be accounted for.

Solution:

- **Class level:** Advance club's member count by each new member.
- **Instance level:** New members receive name and current member count plus 1.

Step 1: Implementing club member names.

```
public class ClubMember {  
    final private String name;  
  
    public ClubMember(final String name) {  
        this.name = name;  
    }  
    public String toString() {  
        return "Member " + name;  
    }  
}
```

Showing membership info.

```
final ClubMember  
    john = new ClubMember("John"),  
    karen = new ClubMember("Karen");  
  
System.out.println(john.toString());  
System.out.println(karen.toString());
```

```
Member John  
Member Karen
```

Step 2: Adding membership numbers.

```
public class ClubMember {  
  
    static ❶ private int memberCount = 0;  
  
    final private int memberNumber; ❷  
    final private String name;  
  
    public ClubMember(final String name) {  
        this.name = name;  
        memberNumber = ++memberCount; ❸  
    }  
    public String toString() {  
        return "Member " + name + ", member number " + memberNumber ❹;  
    }  
}
```

Showing membership numbers.

```
final ClubMember  
    john = new ClubMember("John"),  
    karen = new ClubMember("Karen");  
  
System.out.println(john); ❶ // toString() is being  
System.out.println(karen); // called implicitly
```

```
Member John, member number 1  
Member Karen, member number 2
```

Member creation steps

```
public class ClubMember {
```

```
    static private int memberCount = 0;
```

```
    final private int memberNumber;
```

```
    final private String name;
```

```
    public ClubMember(  
        final String name) {
```

```
        this.name = name;
```

```
        memberNumber = ++memberCount;
```

```
    }
```

```
}
```

2

```
new ClubMember("John")
```

"John"
1

```
new ClubMember("Karen")
```

"Karen"
2

Accessing the club's overall member count?

```
public class ClubMember {  
    static private int memberCount = 0;  
    ...  
    public ClubMember(final String name) {  
        this.name = name;  
        memberNumber = ++memberCount;  
    }  
    static public int getMemberCount() ❶ {  
        return memberCount; ❷  
    }  
    ...  
}
```

Accessing the club's member count

```
final ClubMember
    john = new ClubMember("John"),
    karen = new ClubMember("Karen"),
    petra = new ClubMember("Petra");

System.out.println(karen.toString());

System.out.println("Club's member count
    + ClubMember.getMemberCount());
// Good: Prevent tampering memberCount
// variable.
```

Member Karen, member number 2
Club's member count:3

Syntax accessing class members

Class Variable

```
{class name}.{variableName}
```

Class Method

```
{class name}.{methodName} ( [parameter] )
```

static / non-static wrap up

```
public class X {  
    int a; ①  
    static int b; ②  
}
```

- ① Variable a defined **once per instance** of class X.
 - ② Variable b defined **once per class X**.
-

Finally understanding `System.out.println`

¹ ² ³
`System.out.print(...)`

- 1 Class `System` in package `java.lang`.
- 2 Class variable (static) `out` of type `PrintStream` in class `System`
- 3 One of 9 overloaded methods in class `PrintStream`.

Followup exercises

- 105. Class vs. instance
- 106. Distinguishing leap- and non-leap years
- 107. A method for printing square numbers using `for`, `while` and `do ... while`
- 108. Nicely formatting sine values.
- 109. Extending our interest calculator
- 110. Integer value considerations.
- 111. Programmer's favourite expression
- 112. Lotteries revisited
- 113. Finding the greatest common divisor of two integer values

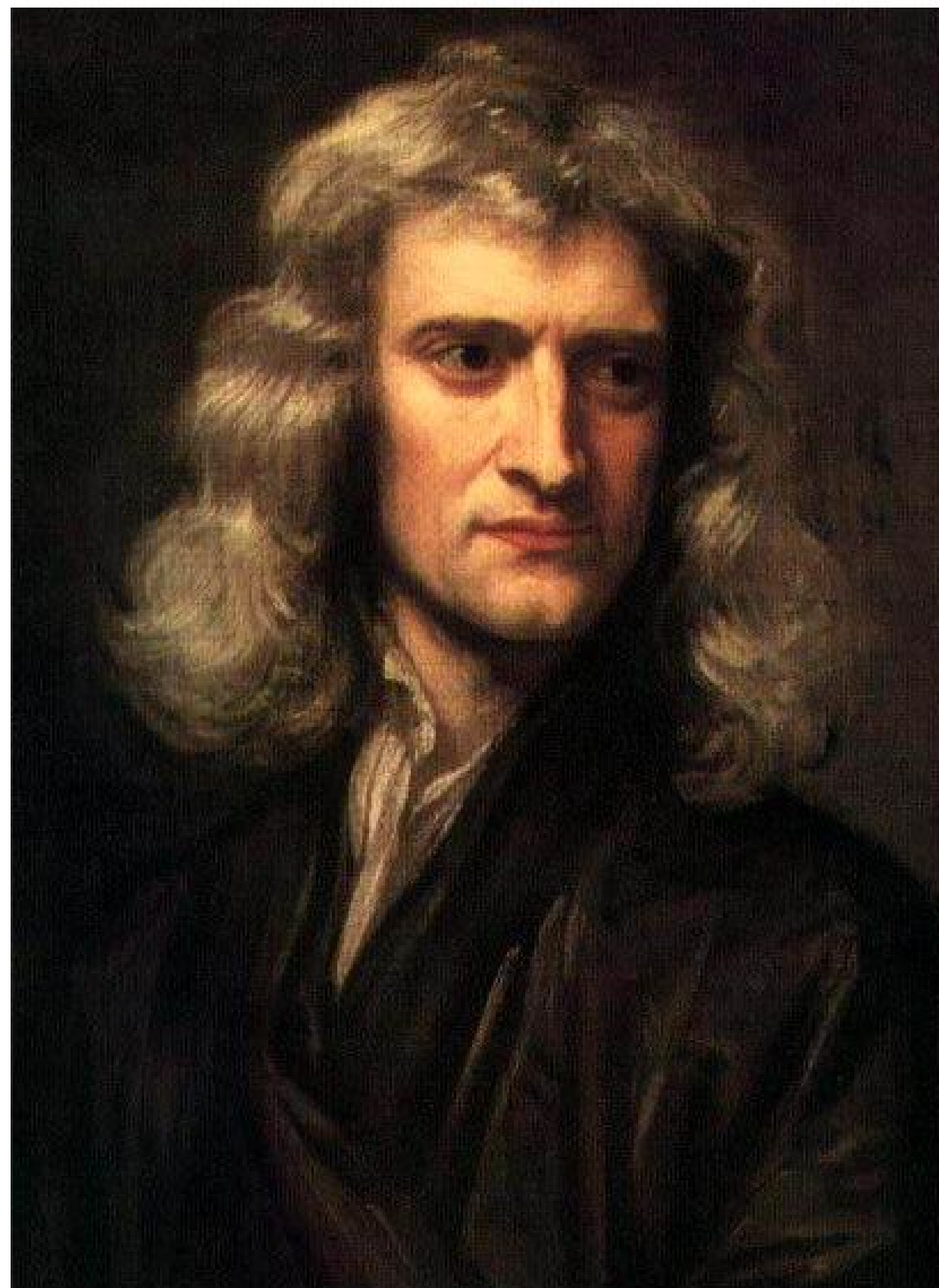
Objects and Classes

⇒ Maven project dependencies

Cross project dependencies.

Repositories and project artifacts.

Newton's letter to Robert Hooke

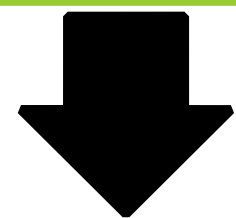


»If I have seen further
it is by standing on ye
sholders of Giants«



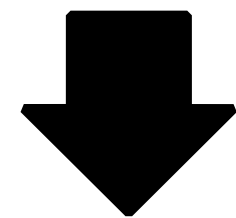
Application execution prerequisites

Application



uses

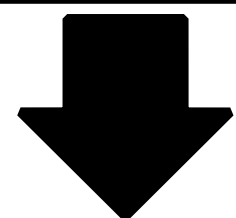
**Java Virtual machine +
libraries**



uses

Userspace tools

glibc, bash, systemd, ...



uses

Linux kernel + modules

~28,000,000 lines of code

Tools:

-Compiler

-Debugger

-IDE

...

Services:

-Network

-Databases

-Web services

-...

Why [Maven](#) project management?

- Automated third-party class import and dependency management
- Executing automated tests
- Complete project lifecycle:
compile, test, execute, package, deploy
- Extensible plugin architecture

Example: Creating PDF using iText7

Source: CreatePdf.java	PDF output
<pre data-bbox="210 688 1537 1360">import com.itextpdf.kernel.pdf.PdfDocument; ... final PdfWriter writer = new PdfWriter("helloWorld.pdf"); final PdfDocument pdf = new PdfDocument(writer); final Document document = new Document(pdf, PageSize.A8.rotate()); document.add(new Paragraph("Hello World!"). setFontColor(ColorConstants.MAGENTA). setFontSize(20)); document.close();</pre>	Hello World!

Maven iText library pom.xml definition



IText Core » 8.0.3

A Free Java-PDF library

License	AGPL 3.0
Tags	pdf
Organization	Apryse Group NV
HomePage	https://itextpdf.com/
Date	Feb 07, 2024
Files	pom (4 KB) View All
Repositories	Central
Ranking	#232395 in MvnRepository (See Top Artifacts)
Used By	1 artifacts

[Maven](#)[Gradle](#)[Gradle \(Short\)](#)[Gradle \(Kotlin\)](#)[SBT](#)[Ivy](#)[Grape](#)[Leiningen](#)[Buildr](#)

```
<dependency>
  <groupId>com.itextpdf</groupId>
  <artifactId>itext-core</artifactId>
  <version>8.0.3</version>
  <type>pom</type>
</dependency>
```

```
<project ...>
  ...
  <dependencies>
    <dependency>
      <groupId>com.itextpdf</groupId>
      <artifactId>itext-core</artifactId>
      <version>8.0.3</version>
      <type>pom</type>
    </dependency>
  </dependencies>
  ...
</project>
```

Itext transitive dependencies

```
mvn dependency:tree
...
--- dependency:3.6.1:tree (default-cli) @ hellopdf ---
de.hdm_stuttgart.mi:hellopdf:jar:0.9
\-- com.itextpdf:itext-core:pom:8.0.2:compile
    +- com.itextpdf:barcodes:jar:8.0.2:compile
    |   \-- org.slf4j:slf4j-api:jar:1.7.36:compile
    +- com.itextpdf:font-asian:jar:8.0.2:compile
    +- com.itextpdf:forms:jar:8.0.2:compile
    +- com.itextpdf:hyph:jar:8.0.2:compile
    +- com.itextpdf:io:jar:8.0.2:compile
    |   \-- com.itextpdf:commons:jar:8.0.2:compile
    +- com.itextpdf:kernel:jar:8.0.2:compile ...
```

Class location in iText library

The screenshot shows an IDE window titled "HelloPdf - pom.xml (hellopdf)". The menu bar includes File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, Git, Window, and Help. The toolbar contains icons for file operations, a "CreatePdf" button, and Git status indicators. The left sidebar shows the "Project" view with "External Libraries" expanded, listing various Maven dependencies from com.itextpdf. The main editor displays the "pom.xml" file with the following content:

```
19
20 <dependencies>
21   <dependency>
22     <groupId>com.itextpdf</groupId>
23     <artifactId>itext7-core</artifactId>
24     <version>8.0.2</version>
25     <type>pom</type>
26   </dependency>
27
28 </dependencies>
29
```

The line containing the version "8.0.2" is highlighted in yellow. The breadcrumb at the bottom of the editor reads: "project > dependencies > dependency > version". The bottom status bar shows "Run: CreatePdf" and a message: "// Build completed successfully in 774 ms (11 minutes ago)". The right sidebar contains icons for Notifications, Database, ScView, and Maven.

Class location in iText library

The screenshot shows an IDE window titled "HelloPdf - CreatePdf.java". The left sidebar displays the project structure for "HelloPdf" at "src/main/java/de/hdm_stuttgart/mi/CreatePdf". The "Maven: com.itextpdf:kernel:8.0.2" dependency is expanded, showing the "kernel-8.0.2.jar library root" and its sub-packages. The "colors" package is expanded, and "ColorConstants" is circled in red. A red arrow points from this circled class to the corresponding import statement in the code editor: `import com.itextpdf.kernel.colors.ColorConstants;`. The code editor also shows other imports: `import com.itextpdf.kernel.geom.PageSize;`, `import com.itextpdf.kernel.pdf.PdfWriter;`, `import com.itextpdf.layout.Document;`, `import com.itextpdf.layout.element.Paragraph;`, and `import java.io.FileNotFoundException;`. The bottom status bar indicates the build completed successfully in 774 ms (16 minutes ago).

Project Structure:

- Maven: com.itextpdf:io:8.0.2
- Maven: com.itextpdf:kernel:8.0.2
 - kernel-8.0.2.jar library root
 - com.itextpdf.kernel
 - actions
 - colors
 - gradients
 - CalGray
 - CalRgb
 - Color
 - ColorConstants**
 - DeviceCmyk
 - DeviceGray
 - DeviceN

Code Editor (CreatePdf.java):

```
4
5 import com.itextpdf.kernel.colors.ColorConstants;
6
7 import com.itextpdf.kernel.geom.PageSize;
8
9 import com.itextpdf.kernel.pdf.PdfWriter;
10
11 import com.itextpdf.layout.Document;
12
13 import com.itextpdf.layout.element.Paragraph;
14
15 import java.io.FileNotFoundException;
```

Run: CreatePdf

Build completed successfully in 774 ms (16 minutes ago)

Maven repositories

CDN

Maven Central

Company local

Sonatype Nexus, e.g. MI Maven repository

114. Dealing with IBAN numbers

Maven archetypes

- Blueprints for projects.
- Based on the `pom.xml` file:
Project **O**bject **M**odel
- Initial project creation.
- **CLI** and **IDE** support.
- Extensibility by archetype catalogs.

```
mvn archetype:generate
[INFO] Scanning for projects...
...
1: remote -> am.ik.archetype:elm-spring-boot-blank-archetype ...
2: remote -> am.ik.archetype:graalvm-blank-archetype ...
3: remote -> am.ik.archetype:graalvm-springmvc-blank-archetype ...
...
2083: remote -> org.apache.maven.archetypes:maven-archetype-quickstart...
...
3330: remote -> za.co.absa.hyperdrive:component-archetype_2.12 (-)
Choose a number or apply filter ...
```

Project “lottery” depending on “common”

Project »common«

```
public class Helper {  
    static public long  
        factorial(int n) {  
        long result = 1;  
        for (int i=2;i<=n;i++){  
            result *= i;  
        }  
        return result;  
    }  
}
```

uses



Project »lottery«

```
a = Helper.factorial(6);
```

Providing project “common”

Helper.java

```
package de.hdm_stuttgart.common;
public class Helper {
    static public long
        factorial(int n) {
        long result = 1;
        for (int i = 2;
            i <= n; i++) {
            result *= i;
        }
        return result;
    }
}
```

pom.xml

```
<project xmlns="http://maven.apache.org...>
    ...
    <groupId>de.hdm-stuttgart</groupId>
    <artifactId>common</artifactId>
    <version>1.0</version>
    ...
</project>
```

Publish project “Common”’s library

```
goik@goiki Helper> mvn install
...
[INFO] Building common 1.0
...
[INFO] Installing .../target/common-1.0.jar to
/home/goik/.m2/repository/de/hdm-stuttgart/common/1.0/common-1.0.jar
```

Content of archive common-1.0.jar

```
goik@goiki tmp> unzip .../de/hdm-stuttgart/common/1.0/common-1.0.jar
Archive:  /home/goik/.m2/repository/de/hdm-stuttgart/common/1.0/common-1.0.jar
  ...
  creating: de/
  creating: de/hdm-stuttgart/
  creating: de/hdm-stuttgart/common/
  inflating: de/hdm-stuttgart/common/Helper.class
  creating: META-INF/maven/
  creating: META-INF/maven/de.hdm-stuttgart/
  creating: META-INF/maven/de.hdm-stuttgart/common/
  inflating: META-INF/maven/de.hdm-stuttgart/common/pom.xml
  inflating: META-INF/maven/de.hdm-stuttgart/common/pom.properties
  ...
  inflating: org/apache/logging/log4j/core/AbstractLifecycle.class  ...
```

Consuming project “lottery”

```
<project ...>
  ...
  <groupId>de.hdm-stuttgart</groupId>
  <artifactId>lottery</artifactId>
  <version>0.9</version>

  <dependencies>
    <dependency>
      <groupId>de.hdm-stuttgart</groupId>
      <artifactId>common</artifactId>
      <version>1.0</version>
    </dependency>
  </dependencies>
  ...
</project>
```

External libraries view

The screenshot displays an IDE interface with the 'External Libraries' view on the left and a 'pom.xml' file open in the editor on the right.

External Libraries View (Left):

- Project
- External Libraries
 - < 17 > /usr/lib/jvm/java-21
 - Maven: de.hdm-stuttgart:common:1.0
 - common-1.0.jar library root
 - de.hdm_stuttgart.common
 - Helper
 - META-INF
 - Maven: junit:junit:4.13.1
 - Maven: org.hamcrest:hamcrest-core:1.3
 - PHP
 - Scratches and Consoles

Editor (Right):

File: pom.xml (lottery)

```
1 <project xmlns="http://maven.apache.org/POM_4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM_4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
16
17 <dependencies>
18   <dependency>
19     <groupId>de.hdm-stuttgart</groupId>
20     <artifactId>common</artifactId>
21     <version>1.0</version>
22   </dependency>
23
24   <dependency>
25     <groupId>junit</groupId>
```

Bottom status bar: Lottery > pom.xml 17:17 LF UTF-8 2 spaces*

Using `Helper.factorial(...)` computing $\binom{n}{k} = \frac{n!}{k! (n - k)!}$

```
static public long binomial(int n, int k) {  
    return (Helper.factorial(n) / Helper.factorial(k)  
           / Helper.factorial(n - k));  
}  
  
public static void main(String[] args) {  
    System.out.println("There are " + binomial(5, 2) +  
        " ways to draw 2 out of 5 numbers");  
  
    System.out.println("There are " + binomial(49, 6) +  
        " ways to draw 6 out of 49 numbers");  
}
```

Maven artifact dependency.

```
public class Helper {  
    static public long factorial(int n) {  
        ... return result;  
    }  
}
```

```
jar tf ~/.m2/repository/de/hdm-stuttgart/...  
...common/1.0/common-1.0.jar...  
de/.../common/Helper.class
```

```
<project ...>...  
  <groupId>de.hdm-stuttgart</groupId>  
  <artifactId>common</artifactId>  
  <version>1.0</version> ...  
</project>
```

```
static public long binomial(int n, int k) {  
    return (Helper.factorial(n) /  
            Helper.factorial(k) /  
            Helper.factorial(n - k));  
}
```

```
<project ...> ...  
  <groupId>de.hdm-stuttgart</groupId>  
  <artifactId>lottery</artifactId>  
  ...
```

```
<dependencies>  
  <dependency>  
    <groupId>de.hdm-stuttgart</groupId>  
    <artifactId>common</artifactId>  
    <version>1.0</version>  
  </dependency> ...
```

mvn install

mvn compile

Followup exercises

115. Cancelling fractions

116. Dealing with local Maven dependencies

Overview

Objects and Classes

- ⇒ Maven project dependencies
 - ⇒ Maven command line usage

Using the MI Sd1 project template

- Download file `mavenTemplate.zip` from [here](#).
- Extract `mavenTemplate.zip` to folder `template`.
- Optional: Edit `template/pom.xml` reflecting your project needs i.e. `<groupId>` and related.
- Optional: Import your project in [IntelliJ IDEA](#).

CLI example

```
mvn --batch-mode -e archetype:generate \  
-DgroupId=de.hdm_stuttgart.mi.sd1 -DartifactId=second -Dversion=0.9 \  
-DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4  
  
[INFO] Scanning for projects...  
    ...  
[INFO] BUILD SUCCESS ...
```

See [artifact reference](#).

Supplementary MI [Maven](#) archetypes

- MI [nexus repository server](#)
- Configuration:

```
mkdir -p ~/.m2 ①  
nano ~/.m2/settings.xml ②
```

CLI testing mi-maven-archetype-quickstart

```
mvn --batch-mode -e archetype:generate \  
  -DgroupId=de.hdm_stuttgart.mi.sd1 -DartifactId=second -Dversion=0.9 \  
  -DarchetypeGroupId=de.hdm_stuttgart.mi -DarchetypeArtifactId=mi-maven-archetype-quickstart -DarchetypeVersion=2.3  
[INFO] Error stacktraces are turned on.  
[INFO] Scanning for projects...  
    ...  
[INFO] BUILD SUCCESS ...
```

CLI archetype details

```
mvn --batch-mode ❶ -e archetype:generate ❷ \  
    \  
-DarchetypeGroupId=de.hdm_stuttgart.mi ❸ \  
-DarchetypeArtifactId=mi-maven-archetype-quickstart \  
-DarchetypeVersion=2.3 \  
    \  
-DgroupId=de.hdm_stuttgart.mi.sd1 ❹ \  
-DartifactId=first \  
-Dversion=0.9
```

Generated project layout

```
> cd first          # Enter project directory
> find . -type f    # Search recursively for files
./pom.xml ❶
./src/main/java/de/hdm_stuttgart/mi/sd1/HighlightSample.java
./src/main/java/de/hdm_stuttgart/mi/sd1/Statistics.java
./src/main/java/de/hdm_stuttgart/mi/sd1/App.java ❷
./src/main/resources/log4j2.xml
./src/test/java/de/hdm_stuttgart/mi/sd1/AppTest.java
./Readme.md
```

117. DNS inflicted groupId / package names clashes

Maven compile

```
> mvn compile
[INFO] Scanning for projects...
...
[INFO] Building first 0.9
...
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to /ma/goik/first/target/classes
[INFO] -----
[INFO] BUILD SUCCESS
```

Compilation file view

```
> find target/classes -type f  
./target/classes/de/hdm_stuttgart/mi/sd1/App.class  
...
```

Execution

```
> cd target/classes ❶  
> java de.hdm_stuttgart.mi.sd1.App ❷  
Hi there, let's have  
fun learning Java! ❸
```

❶ Change to base directory containing compiled **Java™** classes.

❷ Application execution. Note:

Our App class is being prefixed by the package name `de.hdm_stuttgart.mi.sd1` defined by the **groupId** parameter in [Figure 281, “CLI archetype details”](#).

❸ The expected output result.

Note

Executing this particular class requires a configuration in our project's `pom.xml` file:

```
...  
<archive>  
  <manifest>  
    <mainClass>de.hdm_stuttgart.mi.sd1.test.ShowReachedPoints</mainClass>  
  </manifest>  
</archive> ...
```

Maven package

```
> mvn package
...
T E S T S
...
Tests run: 1, Failures: 0, Errors: 0, Skipped: 0
...
[INFO] Building jar: /ma/goik/first/target/first-0.9.jar
...
[INFO] Replacing /ma/goik/first/target/first-0.9.jar with
        /ma/goik/first/target/first-0.9-shaded.jar
...
[INFO] BUILD SUCCESS
```

Executing **Java™** archive `first-0.9.jar`

```
java -jar target/first-0.9.jar  
Hi there, let's have  
fun learning Java!
```

Remark: This will execute `HelloWorld.class` being contained in `first-0.9.jar`.

118. Details on execution

Maven javadoc:javadoc

```
> mvn javadoc:javadoc
[INFO] Scanning for projects...
...
Generating /ma/goik/First/target/site/apidocs/allclasses-noframe.html...
Generating /ma/goik/First/target/site/apidocs/index.html...
Generating /ma/goik/First/target/site/apidocs/overview-summary.html...
Generating /ma/goik/First/target/site/apidocs/help-doc.html...
```

See e.g. [class String documentation](#).

Maven clean

```
> mvn clean
```

```
...
```

```
[INFO]
```

```
[INFO] --- maven-clean-plugin:2.5:clean (default-clean) @ first ---
```

```
[INFO] Deleting /ma/goik/first/target
```

```
[INFO] -----
```

```
[
```

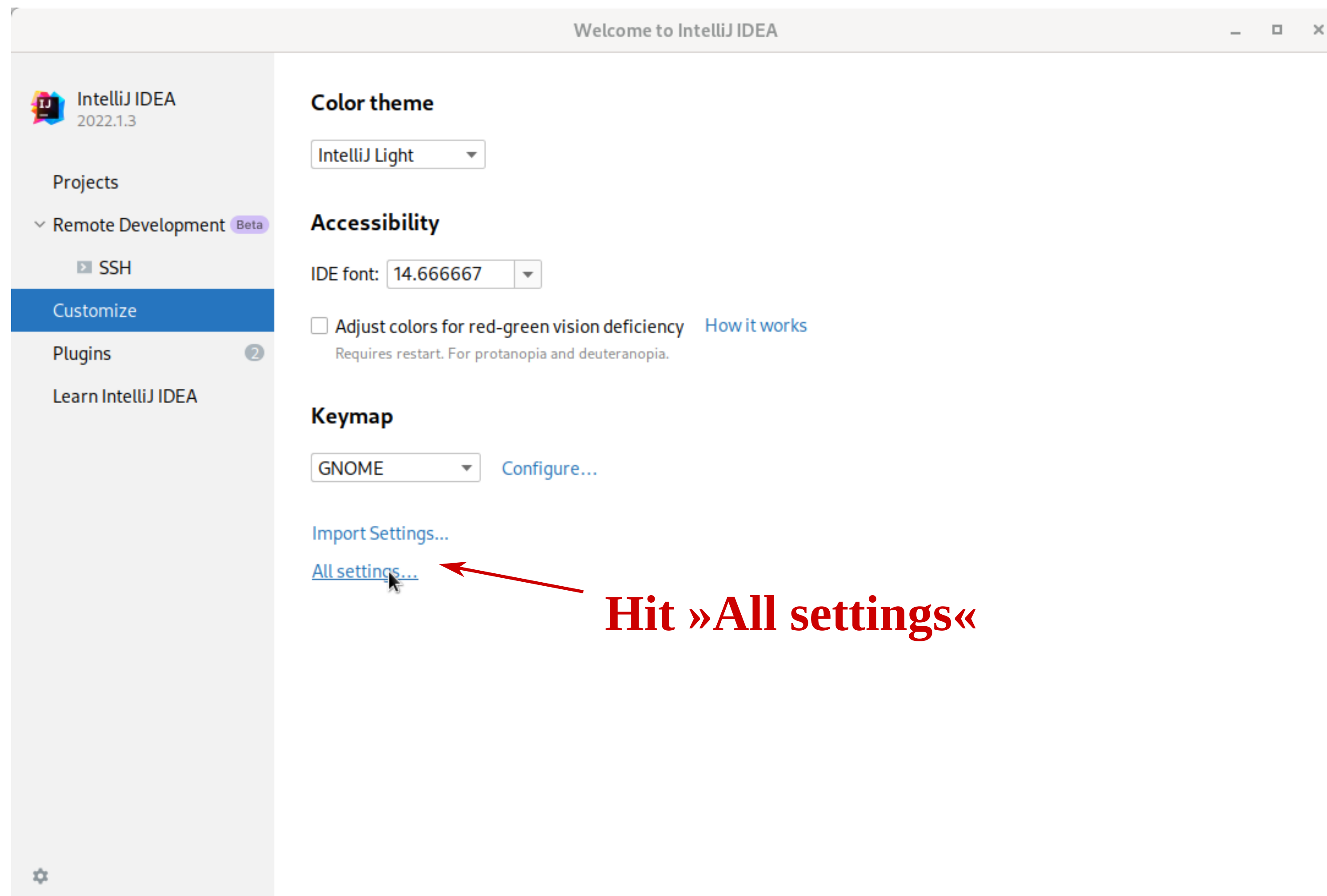
Objects and Classes

- ⇒ Maven project dependencies
 - ⇒ [IntelliJ IDEA](#) on top of Maven

IntelliJ IDEA Maven support

- Built in [Maven](#) support in [IntelliJ IDEA](#).
- MI supplementary archetypes require [MI archetype configuration](#) in `~/.m2/settings.xml`.

Adding MI Maven server



Copy link <https://maven.mi.hdm-stuttgart.de/nexus/repository/mi-maven/archetype-catalog.xml>.

New MI archetype project

1. Select Maven Archetype project

Maven Archetype

Java Enterprise

Spring Initializr

JavaFX

Quarkus

Micronaut

Ktor

Kotlin Multiplatform

Compose Multiplatform

HTML

React

Express

Angular

IDE Plugin

Android

Flask

FastAPI

Django

Composer Package Project

2. Your project's local filesystem location

3. Select MI Archetype server

4. Select desired archetype / version

4. Your project's »coordinates«

5. Hit »Create«

Name: start

Location: ~/IdeaProjects/Start

Project will be created in: ~/IdeaProjects/Start/start

☐ Create Git repository

JDK: 17 version 17.0.3

Catalog: HdM MI extra archetypes

Archetype: de.hdm_stuttgart.mi:mi-maven-archetype-quickstart

Version: 2.1

Additional Properties

Advanced Settings

GroupId: de.hdm_stuttgart.mi.sd1

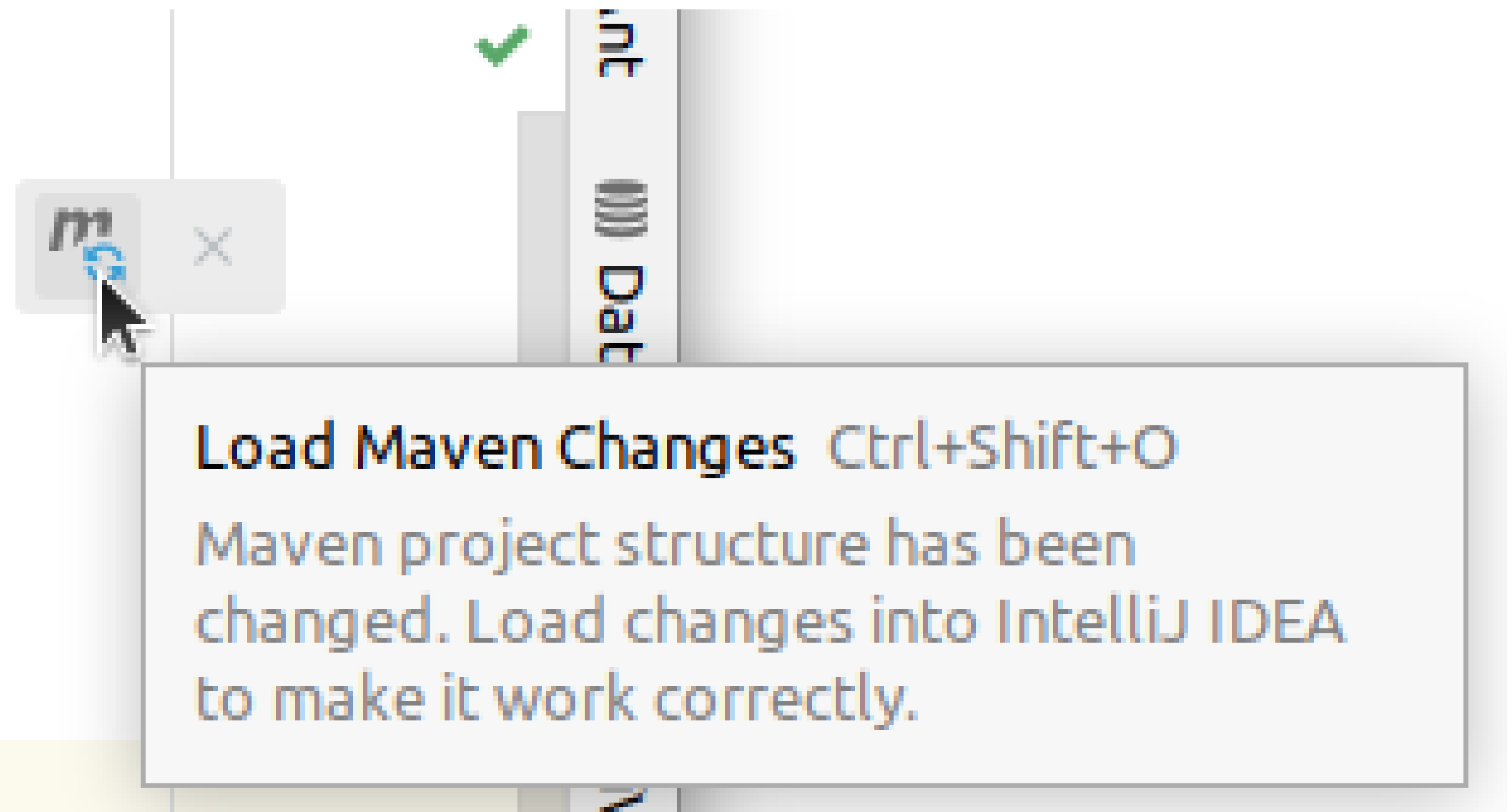
ArtifactId: start

Version: 1.0-SNAPSHOT

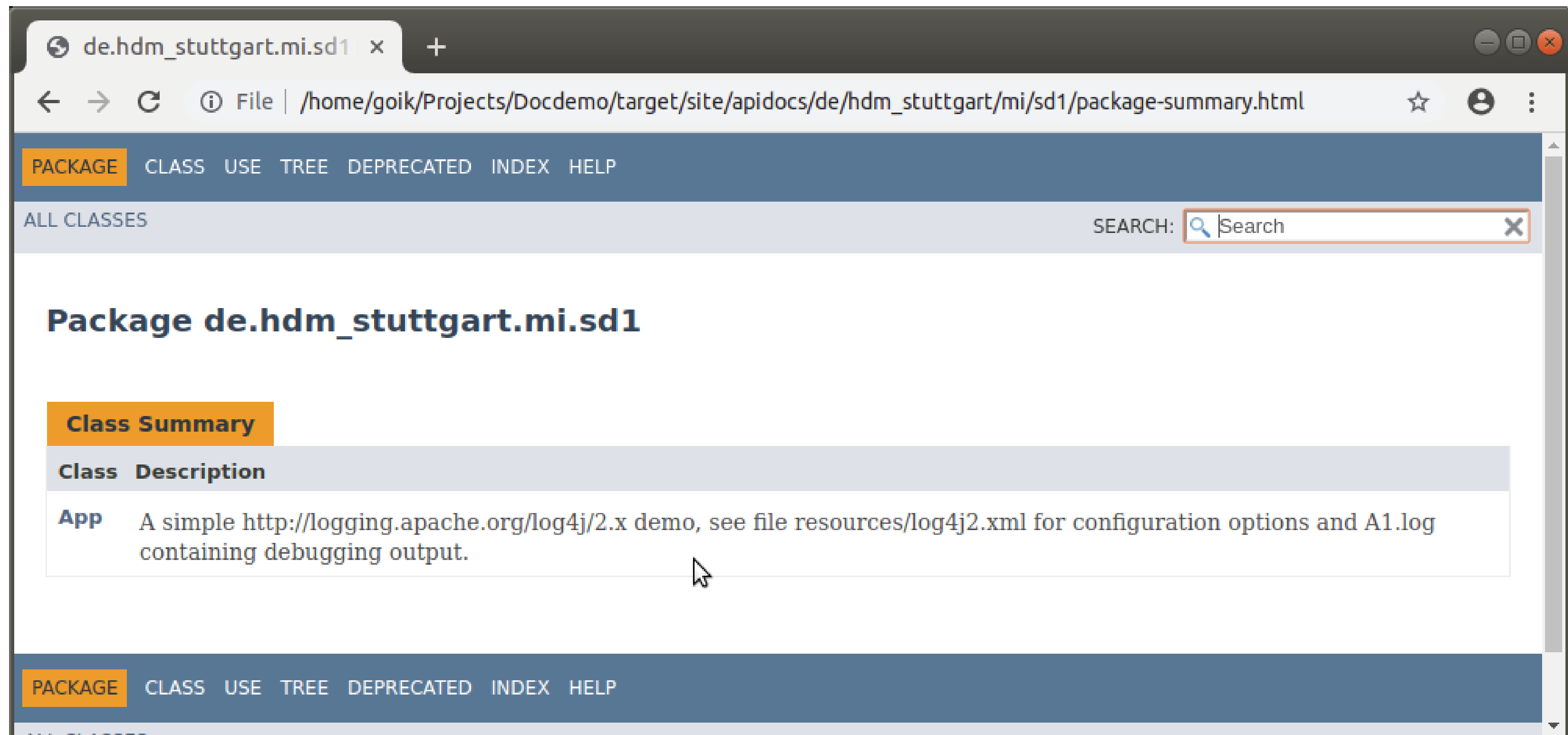
Create Cancel

pom.xml content changes

```
instance"  
1-4.0.0.xsd">
```



- Your **Maven** project depends on its pom.xml file.
- Reloading changed pom.xml file.



Followup exercises

- 119. Maximum and absolute value
- 120. Factorial, the direct way
- 121. Factorial, the recursive way
- 122. Binomials, the recursive way
- 123. The exponential $fx = e^x$
- 124. Implementing $\sin (x)$.
- 125. Summing up in a different order.

Objects and Classes

⇒ Unit testing

Test categories.

Why unit testing? Idea of test «driven development».

Required steps.

Recommended reading

- [\[Vogella2016\]](#)
- [\[TutorialsPointJUnit\]](#)

Test categories

- **Unit test:** Test individual methods, classes and packages in isolation.
- **Integration Test:** Test a group of associated components/classes.
- **Acceptance / Functional Test:** Operate on a fully integrated system, testing against the user interface.
- **Regression Test:** Ensure system integrity after (implementation) change.
- **Load test:** Responsiveness vs. system load.

Example: Computing prime numbers

Informal problem specification:

A prime number is a whole number greater than 1 whose only factors are 1 and itself.

Sample values: 2, 3, 5, 7, 11, 13, 17, 23, ...

Unit test principle

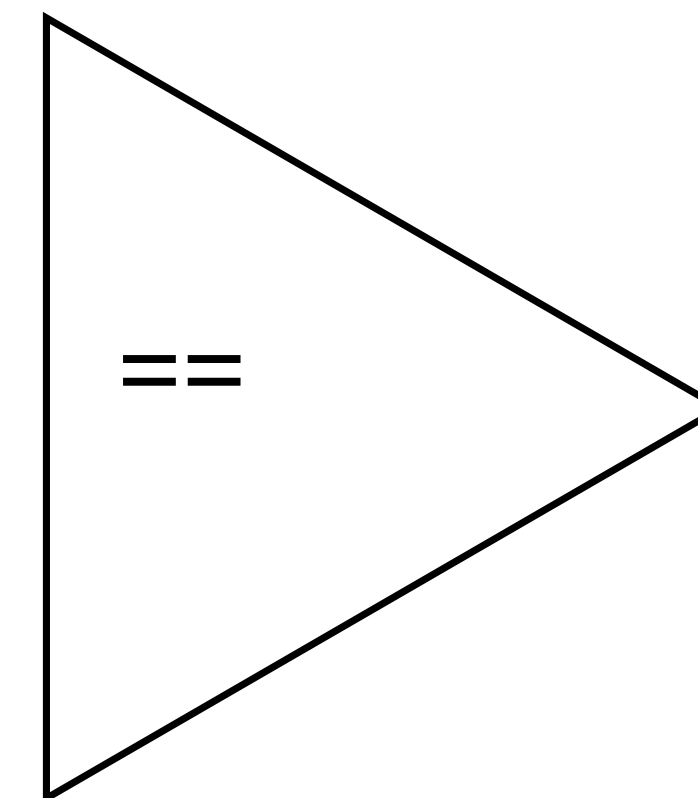
Testing method `isPrime(int n)`

Input: 10



Method
`isPrime(...)`

Computed output
`true`



Expected output: `false`



Failure

Test driven development

First write tests, then implement.

Steps in Unit Testing

1. Specify but not yet implement classes / methods.
2. Write skeleton (dummy) implementations.
3. Write corresponding unit tests.
4. Implement skeleton.
5. Test your implementation.

Overview

Objects and Classes

⇒ Unit testing

⇒ Tests and implementation

Providing both an implementation and related tests possibly revealing coding flaws.

Providing prime number related tests.

Step 1 + 2: Specify method, write skeleton

```
/**
 * Identifying prime numbers.
 */
public class Prime {
    /**
     * Check whether a given integer candidate is prime or not ❶
     * @param candidate A positive integer value
     * @return true if and only if candidate is a prime number.
     */
    public static boolean isPrime(int candidate) {
        return true ❷; //TODO: Dummy value to be implemented correctly
    }
}
```

Execution yet being flawed

```
for (int i = 1; i < 20;i++) {  
    System.out.println(i + " is " + (Prime.isPrime(i) ? " a " : " not a ")  
        + " prime number");  
}
```

```
1 is a prime number  
2 is a prime number  
3 is a prime number  
4 is a prime number  
5 is a prime number  
...
```

Sample test data

Input	Expected output	Input	Expected output
1	false	7	true
2	true	8	false
3	true	9	false
4	false	10	false
5	true	11	true
6	false	12	false

Step 3: Junit based specification test

```
public class PrimeTest {  
  
    @Test ❶ public void test_1_isNotPrime() {  
        Assert.assertFalse(Prime.isPrime(1));  
    }  
    @Test ❶ public void test_2_isPrime() {  
        Assert.assertTrue(Prime.isPrime(2));  
    }  
  
    void someOrdinaryMethod() ❷ {...}  
    ...  
}
```

Junit skeleton test result (Maven CLI)

```
goik@goiki Prime_v01> mvn test
...
Running de.hdm_stuttgart.mi.sd1.PrimeTest
Tests run: 2, Failures: 1, Errors: 0, Skipped: 0,
    Time elapsed: 0.065 sec <<< FAILURE!
...
test_1_isNotPrime(de.hdm_stuttgart.mi.sd1.PrimeTest)
    Time elapsed: 0.001 sec <<< FAILURE!
java.lang.AssertionError
    at org.junit.Assert.fail(Assert.java:86)
    at org.junit.Assert.assertTrue(Assert.java:41)
    at org.junit.Assert.assertFalse(Assert.java:64)
    at org.junit.Assert.assertFalse(Assert.java:74)
...
```

JUnit skeleton test result (IDE)

Run: PrimeTest x

Tests failed: 1, passed: 1 of 2 tests – 24 ms

Test Name	Status	Duration
test_1_isNotPrime	Failed	24 ms
test_2_isPrime	Passed	0 ms

Stack trace for failed test:

```
java.lang.AssertionError <4 internal ca  
at de.hdm_stuttgart.mi.sdl.PrimeTes  
at java.base/jdk.internal.reflect.N  
at java.base/jdk.internal.reflect.N  
at java.base/jdk.internal.reflect.D  
at java.base/java.lang.reflect.Meth
```

Step 3: Providing more prime tests

```
@Test public void test_Primes() {  
    Assert.assertTrue(Prime.isPrime(3));  
    Assert.assertTrue(Prime.isPrime(5));  
    Assert.assertTrue(Prime.isPrime(7));  
    Assert.assertTrue(Prime.isPrime(11));  
    ... }  
@Test public void testOddNonPrimes() {  
    Assert.assertFalse(Prime.isPrime(9));  
    Assert.assertFalse(Prime.isPrime(15));  
    Assert.assertFalse(Prime.isPrime(21));...}
```

Step 3: Prime mass testing

```
@Test public void testEvenNonPrimes() {  
    for (int i = 2; i < 100; i++) {  
        Assert.assertFalse(Prime.isPrime(2 * i));  
    }  
}
```

Step 4: Implement skeleton

```
public static boolean isPrime(int candidate) {  
    for (int i = 2; i < candidate; i++) {  
        if (0 == candidate % i) { // i divides value  
            return false;  
        }  
    }  
    return candidate != 1;  
}
```

Step 5: Testing our first implementation

```
goik@goiki Prime_v01> mvn test
...
-----
T E S T S
-----
Running de.hdm_stuttgart.mi.sd1.PrimeTest
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.055 sec

Results :

Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
```

Objects and Classes

- ⇒ Unit testing

- ⇒ Improving the algorithm

- Try improving the algorithm.

- Introducing an error.

Implementation observation

```
101 / 2 = 50 remainder 1
101 / 3 = 33 remainder 2
101 / 4 = 25 remainder 1
101 / 5 = 20 remainder 1
101 / 6 = 16 remainder 5
101 / 7 = 14 remainder 3
101 / 8 = 12 remainder 5
101 / 9 = 11 remainder 2
101 / 10 = 10 remainder 1
101 / 11 = 9 remainder 2
101 / 12 = 8 remainder 5
101 / 13 = 7 remainder 10
...
```

Changing the implementation

Big performance gain:

```
public static boolean isPrime(int candidate) {  
    for (int i = 2; i * i < candidate; i++) {  
        if (0 == candidate % i) {  
            return false;  
        }  
    }  
    return candidate != 1;  
}
```

Regression test

The screenshot shows the Run window of an IDE. The top bar indicates the project is `de.hdm_stuttgart.mi.sd1` and the test class is `PrimeTest`. The Run tab is active, showing a summary: **Tests failed: 2, passed: 3 of 5 tests – 13 ms**.

The test results list is as follows:

Test Name	Duration	Status
testOddNonPrimes	11 ms	Failed
test_1_isNotPrime	0 ms	Passed
test_Primes	0 ms	Passed
test_2_isPrime	0 ms	Passed
testEvenNonPrimes	2 ms	Failed

The details for the failed test `testEvenNonPrimes` are shown below:

```
java.lang.AssertionError: Testing 4
<3 internal calls>
    at de.hdm_stuttgart.mi.sd1.PrimeTest.testEvenNonPrimes(PrimeTest.java:65)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:566)
```

Systematic error debugging

The screenshot shows an IDE interface with the following components:

- Menu Bar:** File, Edit, View, Navigate, Code, Analyze, Refactor, Build, Run, Tools, VCS, Window, Help.
- Toolbar:** Includes icons for running, debugging, and other IDE functions.
- Project Explorer (Left):** Shows the project structure with folders like 'src', 'test', 'java', and 'de'. The 'PrimeTest' package is selected.
- Code Editor (Center):** Displays the source code for 'PrimeTest.java'. The code includes a Javadoc comment and a method 'isPrime' that checks for prime numbers using a loop and modulo operation. The line 'for (int i = 2; i * i <= value; i++) {' is highlighted.
- Run Window (Bottom):** Shows the execution results of the 'PrimeTest' test. It indicates that all 5 tests passed successfully in 1 ms. The tests listed are: 'testOddNonPrimes', 'test_1_isNotPrime', 'test_Primes', 'test_2_isPrime', and 'testEvenNonPrimes'. The output shows the command used to run the tests and the exit code 0.

Success: Second test failure vanished as well.

Tests passed: 5 (moments ago)

Error correction in detail

```
public static boolean isPrime(int candidate) {  
    //for (int i = 2; i * i < candidate; i++) {  
    for (int i = 2; i * i <= candidate; i++) {  
        if (0 == candidate % i) {  
            return false;  
        }  
    }  
    return candidate != 1;  
}
```

Objects and Classes

- ⇒ Unit testing

- ⇒ Helpful **Junit** methods

- Selected assertion methods.

- Considerations when comparing `float` and `double` values.

Available comparison methods

- `assertEquals([String message],...)`
- `assertArrayEquals([String message],...)`
- `assertFalse([String message],...)`
- `assertNotEquals([String message],...)`
- `assertNull([String message],...)`
- `fail([String message],...)`

Caution comparing float / double !!

```
6  /**
7   * Comparing double values
8   */
9  public class doubleCompareTest {
10     /**
11      * Comparing 3.6 and 3.6 ?
12      */
13     @Test
14     public void test1_3() {
15         Assert.assertEquals( expected: 3.6, actual: 3 * 1.2 );
16     }
17
18
```

'assertEquals(double, double)' is deprecated [more...](#) (Ctrl+F1)

Weird arithmetics?

```
java.lang.AssertionError: Use assertEquals(expected, actual, delta)  
    to compare floating-point numbers  
  
at org.junit.Assert.assertEquals(Assert.java:656)  
at qq.doubleCompareTest.test1_3(doubleCompareTest.java:15)  
...
```

Limited representation precision

```
System.out.println("3.6 - 3 * 1.2 == " + (3.6 - 3 * 1.2));
```

Result:

```
3.6 - 3 * 1.2 == 4.440892098500626E-16
```

Solving the issue

```
public class doubleCompareTest {  
    static final double delta = 1E-15;  
    /**  
     * Comparing 3.6 and 3 * 1.2 within delta's limit  
     */  
    @Test  
    public void test1_3() {  
        Assert.assertEquals(3.6, 3 * 1.2 , delta);  
    }  
}
```

Followup exercises

- 126. Summing up integers to a given limit
- 127. Summing up, the better way

Objects and Classes

- ⇒ Unit testing
 - ⇒ Technical details

Junit Building Blocks: Classes and interfaces

Integration using **Maven**.

The `@Test` annotation

```
public
@interface Test {
    ...
}
```

- `@interface` defining an **annotation**.
- Purpose: Adding meta information for automated detection of test methods.

The `Assert` class

```
public class Assert {  
    public static void assertTrue(  
        String message, boolean condition) { ...}  
  
    public static void assertEquals(  
        long expected, long actual) { ...}  
    ...  
}
```

Importing dependencies

```
<project ...>
...
<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13</version>

    <scope>test</scope>
  </dependency>
  ...
</dependencies>
...
</project>
```

Local: /home/goik/.m2/repository/junit/junit/4.13/junit-4.13.jar

Remote: <https://mvnrepository.com/artifact/junit/junit/4.13>

Dependency archive content

```
> jar -tf junit-4.13.jar
META-INF/
META-INF/MANIFEST.MF
org/
org/junit/
...
org/junit/Assert.class
...
```

Followup exercises

- 128. Turning seconds into weeks, part 2
- 129. Example: A class representing fractions

Objects and Classes

- ⇒ Value types and reference types

- Primitive and value types revisited.

Value vs. reference type variables

Value type: Holding data in associated memory location

- byte
- int
- float
- char
- short
- long
- double
- boolean

Reference type: Holding a reference to an object

Array or class instances i.e. `String`, `java.util.Scanner` etc.

Different behaviour!

Value type	<pre>int a = 1; int b = a; a = 5; System.out.println("a=" + a); System.out.println("b=" + b);</pre>	<pre>a=5 b=1</pre>
Reference type	<pre>StringBuffer r = new StringBuffer("Joe"); StringBuffer s = r; r.append(" Simpson"); System.out.println("r=" + r); System.out.println("s=" + s);</pre>	<pre>r=Joe Simpson s=Joe Simpson</pre>

Value variable Details

```
int a = 1;
```

```
int b = a;
```

```
a = 5;
```

```
System.out.println("a="+a);  
System.out.println("b="+b);
```

Stack

a: 5

b: 1

Heap

No objects
involved

Print result showing:

a=5

b=1

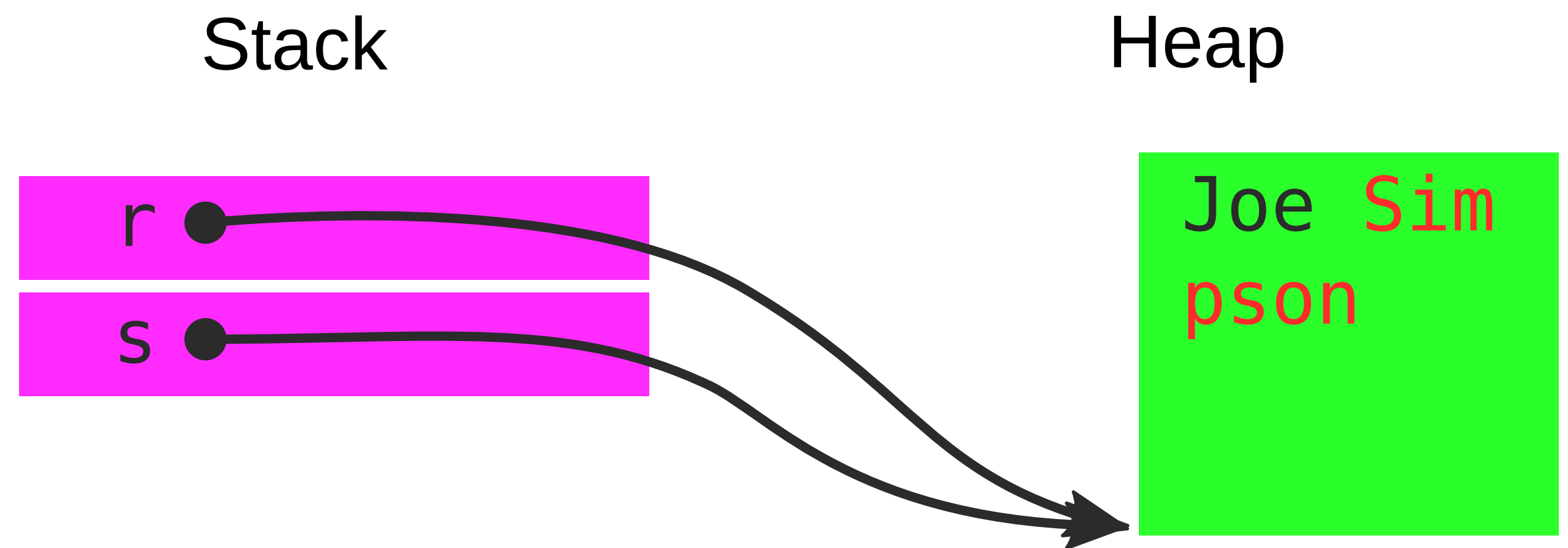
Reference variable Details

```
StringBuffer r =  
    new StringBuffer("Joe");
```

```
StringBuffer s = r;
```

```
r.append(" Simpson");
```

```
System.out.println("r="+r);  
System.out.println("s="+s);
```



Print common StringBuffer instance two times using both variable references a and b in sequence. Result:

Joe Simpson

Joe Simpson

Only “call-by-value” in Java™

```
public static void main(String[] args) {  
    int value = 3;  
    System.out.println(  
        "Before printDuplicateValue: "  
        + value);  
    printDuplicateValue(value);  
    System.out.println(  
        "After printDuplicateValue: "  
        + value);  
}  
static void printDuplicateValue(int n) {  
    n = 2 * n;  
    System.out.println(  
        "printDuplicateValue: " + n);  
}
```

```
Before printDuplicateValue: 3  
printDuplicateValue: 6  
After printDuplicateValue: 3
```

“call-by-value” details

```
...main(String[] args) {  
    int value = 3;  
    ...println("Before  
        printDuplicateValue: "  
        + value);  
    printDuplicateValue(value);  
    ...println("After  
        printDuplicateValue: "  
        + value);  
}  
... printDuplicateValue(int n){  
    n = 2 * n;  
    ...println(  
        "printDuplicateValue: " + n);  
}
```

main(...)

Stack

value: 3

Print variable **value**'s content.
Result: 3

“call-by-reference” for objects?

```
public static void main(String[] args) {  
    StringBuffer buffer = new StringBuffer("My"  
    System.out.println("Before duplicateString:  
        + buffer);  
    duplicateString(buffer);  
    System.out.println("After duplicateString:  
        + buffer);  
}  
static void duplicateString(StringBuffer b) {  
    b.append(b); // Append self  
}
```

Before duplicateString: **My**
After duplicateString: **MyMy**

“call-by-reference” details

```
... main(...) {  
    StringBuffer buffer=  
        new StringBuffer("My");  
    ...println(  
        "Before duplicateString:"  
        + buffer);  
    duplicateString(buffer);  
    ...println(  
        "After duplicateString:"  
        + buffer);  
}  
...duplicateString(  
    StringBuffer b){  
    b.append(b);}
```



Using reference variable `buffer` for
printing `StringBuffer` heap instance.

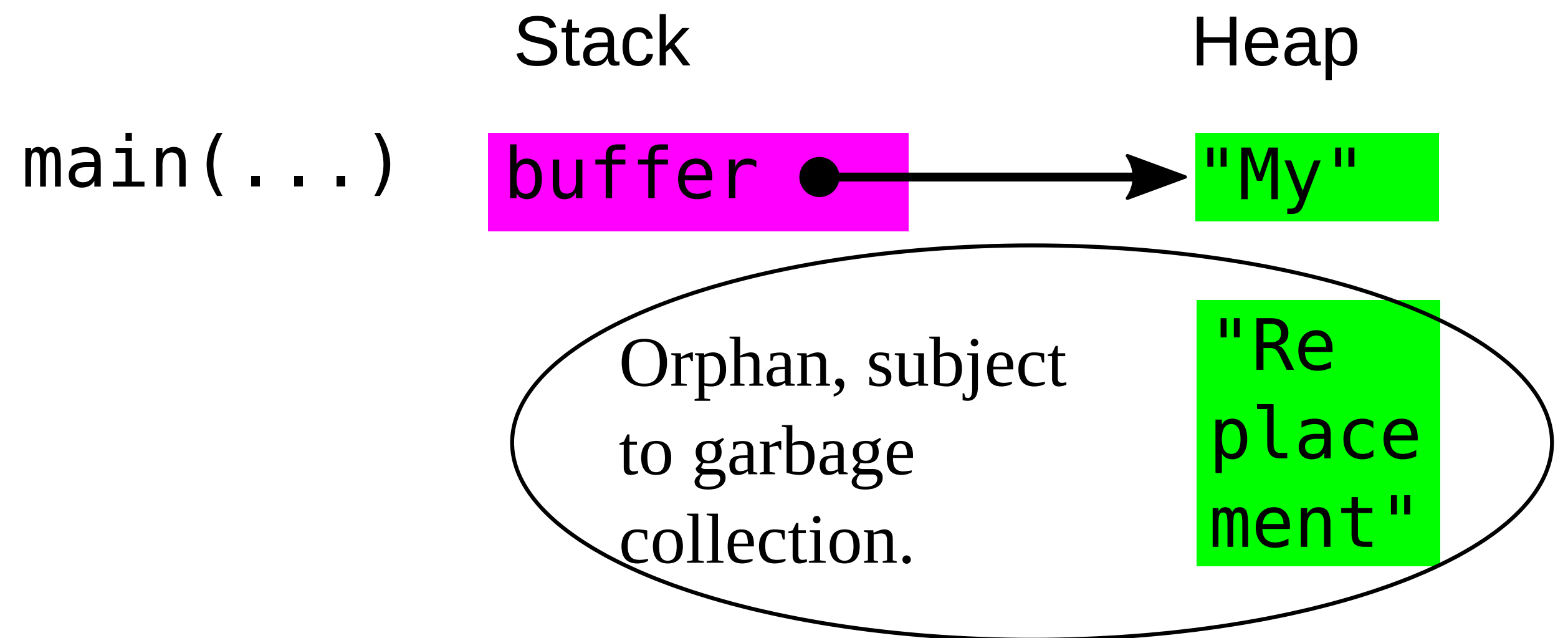
No “call-by-reference” in *Java*[™]!

```
public static void main(String[] args) {  
    StringBuffer buffer = new StringBuffer("My'  
    System.out.println("Before duplicateString:  
        + buffer);  
    replaceString(buffer);  
    System.out.println("After duplicateString:  
        + buffer);  
}  
static void replaceString(StringBuffer b) {  
    b = new StringBuffer("Replacement");  
}
```

Before duplicateString: **My**
After duplicateString: **My**

No “call-by-reference” details

```
... main(...) {  
    StringBuffer buffer  
    =new StringBuffer("My");  
    ...println(  
        "Before: " + buffer);  
    replaceString(buffer);  
    ...println(  
        "After: " + buffer);  
}  
  
replaceString(StringBuffer b){  
    b = new StringBuffer(  
        "Replacement");  
}
```



Print original instance. Result:

"After: My"

Reference to second instance died
along with its defining stack frame
leaving an orphaned heap instance
behind.

C++ reference operator "&"

```
int a = 1;  
  
int &b = a;  
  
cout << a << " : " << b << endl;  
  
a = 5;  
  
cout << a << " : " << b << endl;
```

```
1 : 1  
5 : 5
```

C++ offers “call-by-reference” by virtue of “&”

```
// Passing a reference
// to variable n
void printDuplicateValue(int& n) {
    n = 2 * n;
    cout << "duplicateValue: " << n
        << endl;
}
int main() {
    int value = 3;
    cout << "Before call: "
        << value << endl;
    printDuplicateValue(value);
    cout << "After call: "
        << value << endl;
}
```

```
Before call: 3
duplicateValue: 6
After call: 6
```

C++ “call-by-reference” details

```
... int main() {  
    int value = 3;  
    cout << "Before  
        printDuplicateValue: "  
        << value << endl;  
    printDuplicateValue(value);  
    cout << "Before  
        printDuplicateValue: " ...  
} ... printDuplicateValue(  
    int& n){  
    n = 2 * n;  
    ... "printDuplicateValue: "  
        << n ...;  
}
```

main(...)

Stack

value: 6

Print variable **value**.
Result: 6

Objects and Classes

- ⇒ Method calls, the details

Method calls, behind the scenes.

Stack and heap memory.

Stack frames and value propagation. Using the debugger.

Method calling

```
public class Circle {  
    static final double  
        PI = 3.141592653589793;  
    double r;  
    /** Change a circle's area  
     * @param area The desired new area  
     * @return The circle's new radius */  
    double setArea(final double area ❶) {  
        double val ❷ = area / PI ❸;  
        return ❹ r ❺ = Math.sqrt(val);  
    }  
}
```

❶ Passing arguments.

❷ Defining method local variables.

❸ Accessing class variable.

❹ returning values.

❺ Accessing instance variable.

Three variable scopes

```
public class Circle {  
    static final double  
        PI ❶ = 3.141592653589793;  
  
    double r ❷ ;  
  
    double setArea(final double area ❸) {  
        double val ❸ ...  
        ...  
    }  
}
```

❶ Class scope.

❷ Instance scope

❸ Method scope

Scope lifetimes

<i>Class scope</i> (static)	Application process
<i>Instance scope</i>	Object lifetime: new (. . .) until being garbage collected.
<i>Method scope</i>	Method invocation until return.

Two runtime memory categories

Heap memory

- Allocation of class or array instances:

```
new String()
```

```
new float[200]
```

- De-allocation subject to garbage collection.

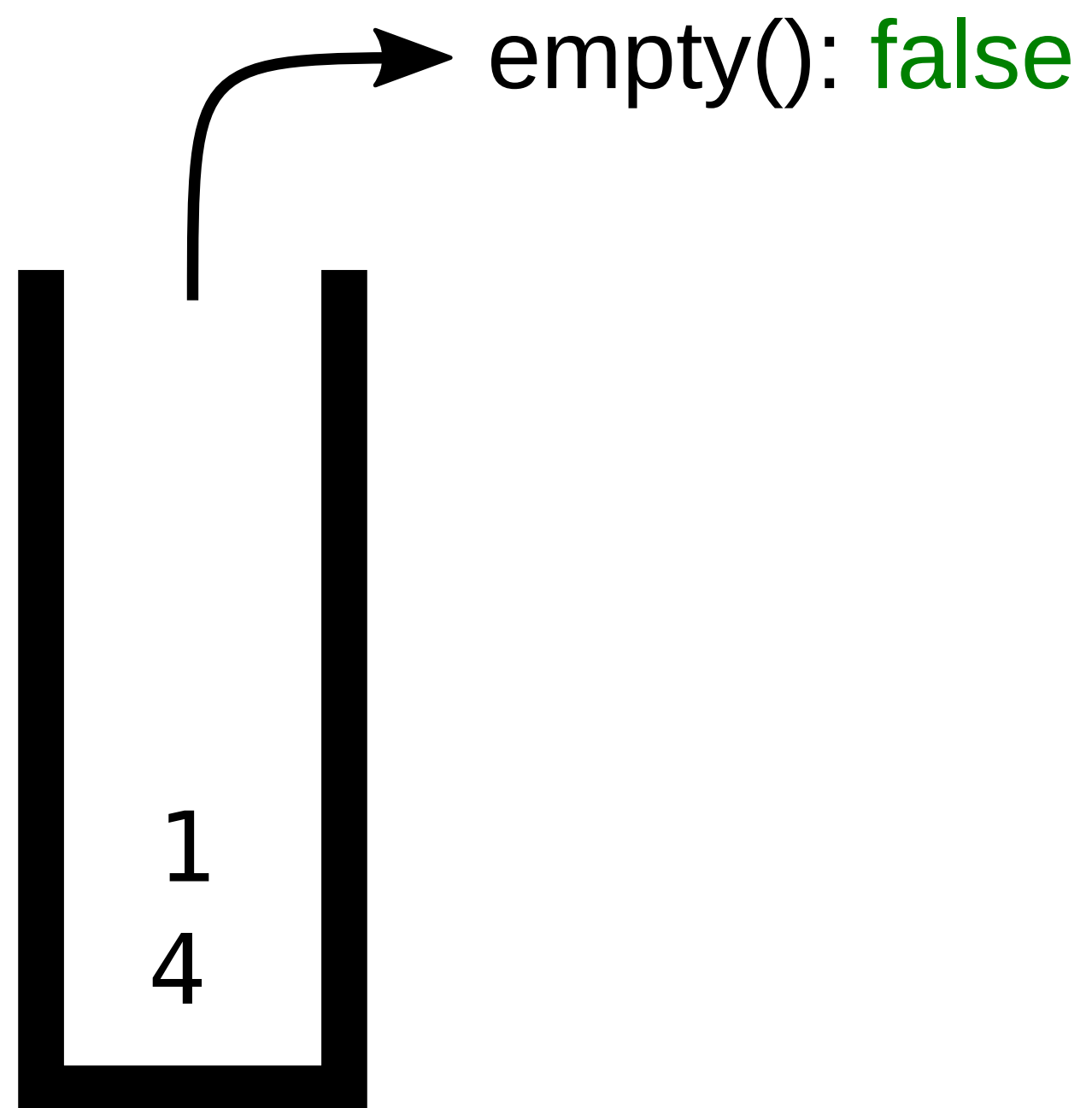
Execution stack

- One instance per process thread.
- Hosting variables (values or references)

Stack: Four operations

Method	Description	Precondition	Access type
<code>push(. . .)</code>	Add object on top	-	Modify
<code>pop()</code>	Read + remove topmost object	Stack not empty	Modify
<code>top()</code>	Read topmost object	Stack not empty	Read only
<code>empty()</code>	true if and only if stack is empty	-	Read only

Example: Storing integer values



Method calling

```
final Stack<Integer> si =  
    new Stack<>();  
si.push(4);  
si.push(1);  
si.push(7);  
System.out.println("Top: " + si.peek()); // top  
while (! si.empty()) {  
    System.out.println("Not empty: " + si.pop());  
}
```

Create new stack of integer values

Call stack trace

```
public class CallStackExample {  
    static double square(double l){  
        return l * l;  
    }  
    static double area(double r) {  
        final double s = square(r);  
        return Math.PI * s;  
    }  
    public ... main(String[] args) {  
        double a = area(2);  
        System.out.println(  
            "Area:" + a);  
    }  
}
```

main(...)



OS

```
2  @  static double square(double l) {  l: 2.0
3  |  return l * l;  l: 2.0
4  |  }
5  @  static double circleArea(double r) {
6  |  final double s = square(r);
7  |  return Math.PI * s;
8  |  }
9  ▶  public static void main(String[] args) {
10 |  double a = circleArea(2);
11 |  System.out.println("Area:" + a);
12 |  }
```

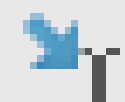
CallStackExample > square()

Debug CallStackExample (1)



Debugger

Console



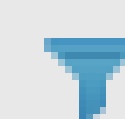
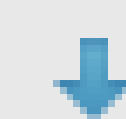
Frames



Variables



"main"@1 in group "main..."



1

square:3, CallStackExample

circleArea:6, CallStackExample

2

l = 2.0

Overview

Objects and Classes

⇒ Enumerations (enum)

Motivation

- Modeling finite sets of discrete states.
- Examples:
 - A room's door: {OPEN, CLOSED}
 - State of matter: {SOLID, LIQUID, GASEOUS}
- No dynamic change of state set.

Objects and Classes

- ⇒ Enumerations (enum)

- ⇒ Enumeration by integer representation

Example: Weekdays Monday till Sunday represented by constant `int` values.

Common pitfall examples.

Insufficient means of language protection against self - inflicted problems.

Weekly offered lectures

```
public class Lecture {  
    public final int dayHeld; /* e.g. to be held on Tuesdays */  
    public final String title; /* e.g. «PHP introduction» */  
    public Lecture(final int dayHeld, final String title) {  
        this.dayHeld = dayHeld;  
        this.title = title;  
    }  
}
```

Weekly offered lectures by simple numbers

Quick and dirty:

```
Class Driver:

final Lecture
    phpIntro      = new Lecture(1 /* Monday */, "PHP introduction"),
    advancedJava = new Lecture(5 /* Friday */, "Advanced Java");
```

Error prone:

- Weeks start on Mondays?
- Index starts with 0 or 1?

Weekdays `int` representation

```
public class Day {  
    static public final int  
        MONDAY    = 1,  
        TUESDAY   = 2,  
        WEDNESDAY = 3,  
        THURSDAY  = 4,  
        FRIDAY    = 5,  
        SATURDAY  = 6,  
        SUNDAY    = 7;  
}
```

Weekly offered lectures using constants

Class `Driver`:

```
final Lecture  
    phpIntro      = new Lecture(Day.MONDAY, "PHP introduction"),  
    advancedJava = new Lecture(Day.FRIDAY, "Advanced Java");
```

Converting index values to day names

```
public class Day {  
    ...  
    public static String getDaysName(final int day) {  
        switch (day) {  
            case MONDAY:    return "Monday";  
            case TUESDAY:   return "Tuesday";  
            case WEDNESDAY: return "Wednesday";  
            case THURSDAY:  return "Thursday";  
            case FRIDAY:    return "Friday";  
            case SATURDAY:  return "Saturday";  
            case SUNDAY:    return "Sunday";  
  
            default:        return "Illegal day's code: " + day;  
        }  
    }  
}
```

Providing lecture info

```
public class Lecture {  
    public final int dayHeld;  
    ...  
  
    public String toString() {  
        return "Lecture «" + title + "» being held each " +  
            Day.getDaysName(dayHeld);  
    }  
}
```

Sample lectures

```
// Class Driver
final Lecture

    phpIntro = new Lecture(
        Day.MONDAY, "PHP introduction"),

    advancedJava = new Lecture(
        Day.FRIDAY, "Advanced Java");

System.out.println(phpIntro);
System.out.println(advancedJava);
```

```
Lecture «PHP introduction»
    being held each Monday
Lecture «Advanced Java»
    being held each Friday
```

Pitfall #1 of 2: Bogus day index value

```
// Class Screwed  
  
final Lecture phpIntro =  
    new Lecture(88, "PHP introduction");  
  
System.out.println(phpIntro);
```

Lecture «PHP introduction» being
held each Illegal day's code: 88

Bad: Not even a compiler warning message!

Pitfall #2 of 2: Method argument order mismatch

```
/**
 * Charge double prices on weekends
 * @param day Day of week
 * @param amount
 * @return the effective amount for
 *         given day of week.
 */
static public int getPrice(
    final int day, final int amount) {
    switch (day) {
        case Day.SATURDAY:
        case Day.SUNDAY: return 2 * amount;

        default: return amount;
    }
}
```

```
// Correct
System.out.println(
    getPrice(Day.SUNDAY, 2));

// Argument mismatch
System.out.println(
    getPrice(2, Day.SUNDAY));
```

```
4
7
```

Bad: Not even a compiler warning message!

Objects and Classes

- ⇒ Enumerations (enum)
 - ⇒ Enumeration by dedicated class

Representing enumeration values by instances gaining type safety.

Partial problem solution.

Enumeration by class instances

Roadmap:

- Define a dedicated enumeration representing class.
- Create exactly one class instance per enumeration value.
- Enumeration value equality comparison by virtue of the == operator.

Class instance per enumeration value

```
public class Day {  
    static public final Day  
        MONDAY    = new Day(),  
        TUESDAY   = new Day(),  
        WEDNESDAY = new Day(),  
        THURSDAY  = new Day(),  
        FRIDAY    = new Day(),  
        SATURDAY  = new Day(),  
        SUNDAY    = new Day();  
}
```

Note: Class without instance attributes.

switch no longer works

Reverting to `if .. else if ...` required 🙄

```
public static String getDaysName(final Day day) {  
    if (MONDAY == day) { // Switch no longer possible, sigh!  
        return "Monday";  
    } else if (TUESDAY == day) {  
        ...  
    } else if (SUNDAY == day) {  
        return "Sunday";  
    } else {  
        return "Illegal day instance: " + day;  
    }  
}
```

Re-writing getPrice()

```
/**
 * Charge double prices on weekends
 * @param day Day of week
 * @param amount
 * @return the effective amount depending on day of week.
 */
static public int getPrice(final Day day, final int amount) {
    if (Day.SATURDAY == day || Day.SUNDAY == day) {
        return 2 * amount;
    } else {
        return amount;
    }
}
```

Compile time argument mismatch error

Preventing **method argument order mismatch**:

```
// Class Driver  
  
// o.K.  
System.out.println(Screwed2.getPrice(Day.SUNDAY, 2));  
  
// Argument mismatch causing compile time type violation error  
System.out.println(Screwed2.getPrice(2, Day.SUNDAY));
```

Pitfall: Creating an undesired instance

Class **Screwed**:

```
final Day PAST_SUNDAY = new Day();  
  
final Lecture phpIntro = new Lecture(  
    PAST_SUNDAY, "PHP introduction");  
  
System.out.println(phpIntro.toString());
```

Lecture «PHP introduction» being
held each **Illegal day instance:**
de.hdm_stuttgart.mi.sd1.
class_wrapper.Day@63961c42

Objects and Classes

- ⇒ Enumerations (enum)

- ⇒ Defining a private constructor

Clearing the mess.

Sad: `switch` still not working.

Define a **private** Day constructor

```
public class Day {  
  
    // Disallow object creation outside class  
  
    private Day() {}  
  
    static public final Day  
        MONDAY    = new Day(),  
        TUESDAY   = new Day(),  
        ...  
        SUNDAY    = new Day();  
}
```

Preventing undesired **Day** instance creation

```
Class Screwed:  
  
Day PAST_SUNDAY = new Day();  
  
Lecture phpIntro = new Lecture(  
    PAST_SUNDAY, "PHP introduction");  
  
System.out.println(phpIntro.toString());
```

Compile time error:

```
'Day()' has private access in  
'de.hdm_stuttgart.mi.sd1.  
class_wrapper_private.Day'
```

Adding a day name attribute

```
public class Day {  
    public final String name;  
  
    private Day(final String name) {  
        this.name = name;  
    }  
    static public final Day  
        MONDAY    = new Day("Monday"),  
    ...  
        SUNDAY    = new Day("Sunday");  
  
    public String toString() { return name; }  
}
```

Objects and Classes

- ⇒ Enumerations (enum)

- ⇒ enum replacing class

Introducing enum as kind of restricted class.

Makes switch working again!

enum Day replacing public class Day

```
public enum Day {  
    MONDAY("Monday"),  
    TUESDAY("Tuesday"),  
    ...  
    SUNDAY("Sunday");  
  
    final String name;  
    Day(final String name) { this.name = name;}  
  
    public String toString() { return name;}  
}
```

switch statements working again

```
public enum Day {  
    ...  
    public static String getItalianDayName(final Day day) {  
        switch (day) {  
            case MONDAY:    return "Lunedì";  
            case TUESDAY:   return "Martedì";  
            ...  
            case SUNDAY:    return "Domenica";  
        }  
        return null; // Actually unreachable, but static  
                     // compiler code analysis is limited  
    }  
}
```

enum constructor being implicitly `private`

```
public enum Day {  
    ...  
    private Day(final String name)  
    { ...
```

Compile time warning:

Modifier 'private' is redundant for enum constructors

```
public enum Day {  
    ...  
    public Day(final String name)  
    { ...
```

Compile time error:

Modifier 'public' not allowed here

Prohibits `enum` external instance creation.

Followup exercises

130. Compass directions

131. Compass direction neighbours

132. Former zodiac examination task

Objects and Classes
→ Using git

Git:

1. A completely ignorant, childish person with no manners.
2. A person who feels justified in their callow behaviour.
3. A pubescent kid who thinks it's totally cool to act like a moron on the internet, only because no one can actually reach through the screen and punch their lights out.

Useful links

- [A recipe for disaster](#)
- [Git tutorial](#)
- [A Practical Introduction to git](#)

Objects and Classes

- ⇒ Using git

- ⇒ Working with **git** locally.

- Creating a git project

- Important commands: git add, git commit, git log, ...

Initialize git project

- `git init`
- Result: Sub folder `.git` containing project's metadata and versioning history

Configure author related data.

- `git config --global user.email "foo@company.com"`
- `git config --global user.name "Helen Foo"`

Adding resources to project index and staging area

- ```
public class Math {

 static public int add(
 final int a, final int b) {
 return a + b;
 }
}
```
- **git status**: Math.java yet unversioned.
- **git add** Math.java
- **git status**: Math.java became versioned.

## Committing change set

- `git commit - -message` "New math class containing single method"
- Result: Commit change set to repository adding given message to project log.

# Project versioning status

```
git status
```

```
On branch master ❶
```

```
No commits yet ❷
```

```
Untracked files: ❸
```

```
(use "git add <file>..." to include in what will be committed)
```

```
 Math.java
```

```
nothing added to commit but untracked files present (use "git add" to track) ❹
```

# Adding a comment

```
public class Math {

 static public int add(
 final int a, final int b)
 return a + b;
 }
}
```

```
public class Math {
 /**
 * Summing two int values.
 * @param a first value.
 * @param b second value.
 * @return The sum of both.
 */
 static public int add(
 final int a, final int b) {
 return a + b;
 }
}
```

# git diff tracing changes

```
diff --git a/Math.java b/Math.java
index 5d72bde..d273b77 100644
--- a/Math.java
+++ b/Math.java
@@ -1,5 +1,10 @@
 public class Math {
-
+ /**
+ * Summing two int values.
+ * @param a first value.
+ * @param b second value.
+ * @return The sum of both.
+ */
 static public int add(
 final int a, final int b) {
 return a + b;
 }
```

Reverting individual file.

```
>git checkout -- ❶ Math.java ❷
>git diff Math.java ❸
>
```

❶ Double dashes '--' disambiguating branch names from file names.

❷ Replace working tree file Math.java by repository master.

❸ No difference working tree to master branch.

Compiling, `Math.class` and `Print.class`.

```
> javac Print.java Math.java # Compilation creating Math.class and Print.class
```

```
> git status
```

```
On branch master
```

```
Your branch is up to date with 'origin/master'.
```

```
Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)
```

```
Math.class
```

```
Print.class
```

```
nothing added to commit but untracked files present (use "git add" to track)
```

`Math.class`, `Print.class` and versioning.

1. `Math.class` and `Print.class` are being generated from `Math.java` and `Print.java`.
2. Rule of thumb: Do not version dependent objects.

Solution: Add a `.gitignore` file to versioning to contain:

```
ignore generated .class files
*.class
```

# Show project's log

```
git log
```

```
commit 0137ccd857a242f4751e36bdbce365c6130c3a32 ① (HEAD -> master)
```

```
Author: Martin Goik <goik@hdm-stuttgart.de>
```

```
Date: Sat May 25 11:56:00 2019 +0200
```

```
 Removing duplicate headline ②
```

```
commit 7f119fac36e02e4c5a7f04f022217b6f744d6e1d ③
```

```
Author: Martin Goik <goik@hdm-stuttgart.de>
```

```
Date: Sat May 25 11:49:52 2019 +0200
```

```
 Project Readme.md ④ ...
```

Switch to an older revision ...

```
git checkout 7f119fac36e02e4c5a7f04f022217b6f744d6e1d
```

Note: checking out '7f119fac36e02e4c5a7f04f022217b6f744d6e1d'.

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using `-b` with the checkout command again. Example:

```
git checkout -b <new-branch-name>
```

HEAD is now at 7f119fa Project README.md

... and forth to current master's HEAD

```
git checkout master
Previous HEAD position was 7f119fa Project Readme.md
Switched to branch 'master'
```

133. git local, DIY

## Objects and Classes

- ⇒ Using git

- ⇒ Shared development with centralized remote.

- Connecting local to remote repository or clone

- push local to remote

- pull from remote to local

# Centralized remote repository

See [multi+remote/shared git](#):

1. [Create empty remote repository](#) on [gitlab.mi.hdm-stuttgart.de](#).
2. Connect local repository to remote
3. [push](#) or [pull](#) content

# Step 1: Create remote repository

← → ↻ 🔒 https://gitlab.mi.hdm-stuttgart.de/projects/new ☆ # 🔁 🌐 🗺️ 🔍 | 👤

**GitLab** Projects ▾ Groups ▾ Activity Milestones Snippets 🕒 + ▾ Search or jump to... 🔍 📄 🔗 📧 ? ▾ 👤 ▾

mes (repository), plan your work (issues), and publish your documentation (wiki), among other things.

All features are enabled for blank projects, from templates, or when importing, but you can disable them afterward in the project settings.

To only use CI/CD features for an external repository, choose **CI/CD for external repo**.

**Tip:** You can also create a project from the command line. [Show command](#)

1

### Project name

Project URL

goik ▾

Project slug

Want to house several dependent projects under the same namespace? [Create a group](#).

### Project description (optional)

Description format

2

### Visibility Level ?

☒  **Private**  
Project access must be granted explicitly to each user.

☐  **Internal**  
The project can be accessed by any logged in user.

☐  **Public**  
The project can be accessed without any authentication.

3

### Initialize repository with a README

Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.

## Step 2: Retrieve remote repository address

**GitIntro**   
Project ID: 2764

 [Add license](#)

### The repository for this project is empty

You can create files directly in GitLab using one of the following options.

[+ New file](#) [+ Add README](#) [+ Add CHANGELOG](#) [+ Add CONTRIBUTING](#)

  Star 0 [Clone](#) 

**2** **Clone with SSH**

`git@gitlab.mi.hdm-stuttgart.` 

**Clone with HTTPS**

`https://gitlab.mi.hdm-stuttg` 

 **Copy URL to clipboard**

### Command line instructions

You can also upload existing files from your computer using the instructions below.

#### **1** Git global setup

```
git config --global user.name "Dr. Martin Goik"
git config --global user.email "goik@hdm-stuttgart.de"
```

## Step 2: Connect to remote repository

```
git remote add origin ① https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git
```

- ① **origin** is an alias for our remote repository `https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git`.

## Step 3: Push local to remote

```
git push --set-upstream origin ❶ master
Username for 'https://gitlab.mi.hdm-stuttgart.de': goik ❷
Password for 'https://goik@gitlab.mi.hdm-stuttgart.de':
Counting objects: 5, done.
Delta compression using up to 6 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 507 bytes | 507.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git
 * [new branch] master -> master ❸
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

## Step 3: Pull remote to local

```
> git pull ❶
Username for 'https://gitlab.mi.hdm-stuttgart.de': goik ❷
Password for 'https://goik@gitlab.mi.hdm-stuttgart.de':
remote: Enumerating objects: 8, done.
remote: Counting objects: 100% (8/8), done.
remote: Compressing objects: 100% (6/6), done.
remote: Total 6 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (6/6), done.
From https://gitlab.mi.hdm-stuttgart.de/goik/gitintro ❸
 733e541..ffff092 master -> origin/master
Updating 733e541..ffff092 ❹
Fast-forward
 Math.java | 10 ++++++-- ❺
1 file changed, 8 insertions(+), 2 deletions(-)
```

## Alternative: Create remote, then clone

1. Create new possibly non-empty project at <https://gitlab.mi.hdm-stuttgart.de>.

2.

```
> git clone https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git
Cloning into 'gitintro'...
Username for 'https://gitlab.mi.hdm-stuttgart.de': goik
Password for 'https://goik@gitlab.mi.hdm-stuttgart.de':
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 5 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (5/5), done.
```

# Overview

## Objects and Classes

- ⇒ Using git

- ⇒ Conflicts

- Line level conflicts

- Manual merge

# Conflicting changes

**User A: Print.java**

```
// Driver class
public class Print {
 public static void
 main(String[] args) {
 System.out.println
 (Math.add(2, 3));
 }
}
```

**User B: Print.java**

```
/**
 * Application entry point
 */
public class Print {
 public static void
 main(String[] args) {
 System.out.println
 (Math.add(2, 3));
 }
}
```

# Commit schedule

| User A                            | User B                                       |
|-----------------------------------|----------------------------------------------|
| Edit: ... <b>Driver class</b> ... | -                                            |
| git commit, git push              | -                                            |
| -                                 | edit: ... <b>Application entry point</b> ... |
| -                                 | git commit                                   |
| -                                 | git push: <b>Fail!</b>                       |

## User B: git push fails

```
>git push ...
To https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git
! [rejected] master -> master (fetch first)
error: failed to push some refs to
 'https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
```

User B: git pull fails as well

```
>git pull
...
From https://gitlab.mi.hdm-stuttgart.de/goik/gitintro
 b003a82..dbbedb0 master -> origin/master
Auto-merging Print.java
CONFLICT (content): Merge conflict in Print.java
Automatic merge failed; fix conflicts and then commit the result.
```

# Merge conflict details

```
>git diff Print.java
diff --cc Print.java
index fc36ae6,7b27edf..0000000
--- a/Print.java
+++ b/Print.java
@@@ -1,6 -1,4 +1,10 @@@
++<<<<<< HEAD
+/**
+ * Application entry point
+ */
+=====
+ // Driver class
++>>>>>> dbbedb0fc29d77beeaada37f2538d78f82bac93
public class Print {
 public static void
 main(String[] args) {
 System.out.println
 (Math.add(2, 3));
 }
}
```

Struggling for resolution

# GIT MERGE



# Merging Print.java manually

```
<<<<<<< HEAD
/**
 * Application entry point
 */
=====
// Driver class
>>>>>> 10cf21c ... 759462c
public class Print {
 public static void
 main(String[] args) {
 System.out.println
 (Math.add(2, 3));
 }
}
```



```
/**
 * Driver class, application entry point
 */
public class Print {
 public static void
 main(String[] args) {
 System.out.println
 (Math.add(2, 3));
 }
}
```

# Commit and push merge

```
>git add Print.java

>git commit --message "Merging changes"
[master 173487a] Merging changes

>git push
...
To https://gitlab.mi.hdm-stuttgart.de/goik/gitintro.git
10cf21c..173487a master -> master
```

134. git distributed, DIY