

Basics of Java Programming

Hendrik Speleers

Basics of Java Programming

- **Overview**
 - Building blocks of a Java program
 - Classes
 - Objects
 - Primitives
 - Methods
 - Memory management
 - Making a (simple) Java program
 - Baby example
 - Bank account system

Basics of Java Programming

- **A Java program**
 - Consists of **classes** (existing ones and/or new ones)
 - Has one class with a **main** method (to start the program)
- **Syntax of a class**
 - Comments and embedded documentation
 - Import from libraries (by default: java.lang.*)
 - Class declaration: collection of **variables** and **methods**
- **Compiling and running**
 - `javac Hello.java`
 - `java Hello`

Basics of Java Programming

- A simple Java program (1)

```
// Hello.java  
  
// Print "Hello, world" to the console  
  
public class Hello {  
  
    public static void main(String[] args) {  
        System.out.println("Hello, world");  
    }  
}
```

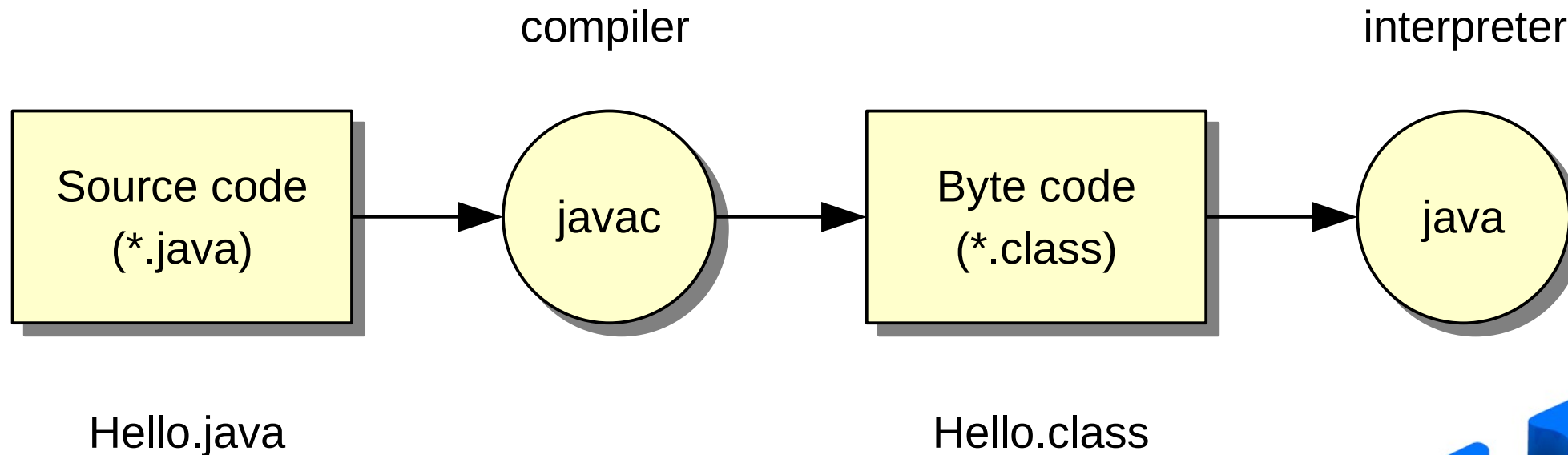
} Comments

} Class declaration

Note: every statement ends with semi-colon ;

Basics of Java Programming

- A simple Java program (1)



Basics of Java Programming

- A simple Java program (2)

```
// HelloDate.java

import java.util.*;

public class HelloDate {
    public static void main(String[] args) {
        System.out.println("Hello, it is");
        Date date = new Date();
        System.out.println(date.toString());
    }
}
```

} Comments

} Import from library

} Class declaration

Note: every statement ends with semi-colon ;

Basics of Java Programming

- **Comments**

- Intended for the reader as documentation
- Two possibilities
 - Multi-line comment between `/*` and `*/`

```
/* This is a comment that  
*  continues across lines  
*/
```

- Single-line comment after `//`

```
// This is a one-line comment
```

Basics of Java Programming

- Declaration of classes

```
<modifiers> class <class name> {  
    <variable declarations>  
    <method declarations>  
}
```

- Collection of **variables** (storage of data) and **methods** (actions on data)
- In our example:
 - Modifiers: public (3 access modifiers: public – private – protected)
 - Name: HelloDate
 - Fields: no class variables
 - Methods: main

Basics of Java Programming

- Declaration of methods

```
<modifiers> <return type> <method name> (<parameters>) {  
    <method body>  
}
```

- In our example:

- Modifiers: public static (it belongs to the class instead of a specific object)
- Return type: void (= no return value)
- Name: main
- Parameters: String args[] (array of strings)

- Exiting method with value

```
return <variable> ;
```

Basics of Java Programming

- **Variables: primitive types**
 - Same syntax and operations as in C++

	Type	Meaning	Memory size
integers {	byte	very small integer (-128,...,127)	8 bits
	short	small integer	16 bits
	int	integer	32 bits
	long	long integer	64 bits
reals {	float	single-precision floating point number	32 bits
	double	double-precision floating point number	64 bits
	char	character (Unicode)	16 bits
	boolean	true or false	

Basics of Java Programming

- **Variables: primitive types**

- Declaration and assignment

```
<data type> <variables> ;
```

```
<variable> = <expression> ;
```

```
final <data type> <variable> = <expression> ;
```

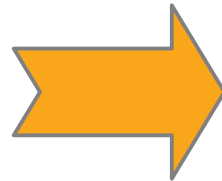
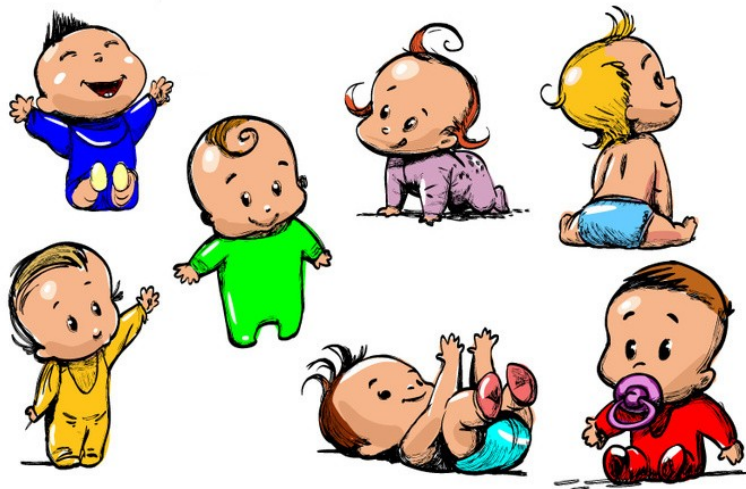
constant variable

- Examples:

```
int i = 10, j;  
j = i + 5;  
final double PI = 3.141592;
```

Basics of Java Programming

- Baby example



Class

Baby

Baby

Baby

Baby

Baby

Baby

Baby

Baby

Objects

Basics of Java Programming

- **Baby example**

- A class for babies containing
 - name
 - sex (m/f)
 - weight (kg)
 - # poops so far
- How to make Baby objects ?

```
public class Baby {  
    String name = "Unknown";  
    boolean isMale = true;  
    double weight = 0.0;  
    int nbPoops = 0;  
  
    void poop() {  
        nbPoops = nbPoops + 1;  
        System.out.println(  
            "Mam, I have pooped."  
            + " Ready the diaper."  
        );  
    }  
}
```

Variables

Methods

Basics of Java Programming

- Declaration and creation of objects

- Creating an object variable (object declaration)

```
<class name> <object name> ;
```

object:
an instance of a class

- Creating an object with the **new** keyword

```
<object name> = new <class name> (<arguments>) ;
```

- Example: creating a **String** object

```
String str = new String("abc");  
String str = "abc";
```

a String “behaves”
like a primitive



Basics of Java Programming

- Initialization of objects

- The **constructor**: a special method with class name

```
<access modifier> <class name> (<parameters>) {  
    <constructor body>  
}
```

- Purpose: giving valid values to the class variables for the specific object
 - No return type
- Default constructor: automatic, when no other constructors

```
public <class name> () {  
}
```

no parameters
empty body

Basics of Java Programming

- **Baby example**
 - Let's update the baby class with constructor and some methods

```
public class Baby {  
    ...  
    Baby(String n, boolean m, double w) {  
        name = n; isMale = m; weight = w;  
    }  
    void sayHi() {  
        System.out.println("Hi, my name is " + name);  
    }  
    void eat(double food) {  
        weight = weight + food;  
    }  
}
```


Basics of Java Programming

- **Using objects**

- Externally accessing a variable + sending a message to an object

```
<object name> . <variable name> ;
```

```
<object name> . <method name> (<arguments>) ;
```

- Example: let's make a baby object

```
Baby david = new Baby("David", true, 4.0);  
System.out.println(david.name);  
david.eat(0.1);  
david.poop();
```

Basics of Java Programming

- **Static types and methods**
 - The **static** keyword implies
 - The variable/method is part of the class declaration
 - It is unique for the class and NOT for each instance (object)
 - Example: keeping track of number of babies made

```
public class Baby {  
    static int nbBabiesMade = 0;  
    Baby(String n, boolean m, double w) {  
        name = n; isMale = m; weight = w;  
        nbBabiesMade = nbBabiesMade + 1;  
    }  
}
```

External access
via class name

Basics of Java Programming

- **Arrays**

- An array is a sequence of elements of same type (primitives / objects)
- Declaration and creation

```
<type>[] <array name> = new <type>[<integer>] ;
```

an array is an object

- Example:

```
Baby[] twin = new Baby[2];  
twin[0] = new Baby("Oliver", true, 4.0);  
twin[1] = new Baby("Olivia", false, 4.0);
```

index of first position
in an array is zero



Basics of Java Programming

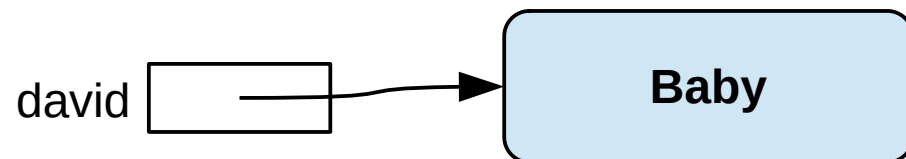
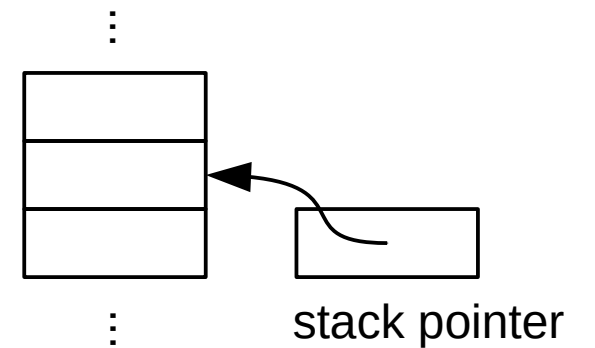
- **Baby example**
 - Let's make a nursery

```
public class Nursery {  
    final int CAPACITY = 25;  
    Baby[] babies = new Baby[CAPACITY];  
    int nbBabies = 0;  
    ...  
    void addBaby(Baby baby) {  
        // Assume: nbBabies < CAPACITY  
        babies[nbBabies] = baby;  
        nbBabies = nbBabies + 1;  
    }  
}
```

Basics of Java Programming

- **Memory management**

- Different places to store data
 - Register: inside the processor, very fast but very limited
 - The stack: in RAM, direct support from processor (stack pointer)
 - The heap: in RAM, general-purpose pool of memory
- Primitive types in the stack
- Object declaration in the stack (a pointer)
Object creation in the heap (with the **new** keyword)



Basics of Java Programming

- **Memory management**
 - Working with primitives

```
int i = 10, j;  
final double PI = 3.141592;
```

i 10

j ?

PI 3.14

Basics of Java Programming

- **Memory management**
 - Working with primitives

```
int i = 10, j;  
final double PI = 3.141592;  
j = i;  
i = 5;
```

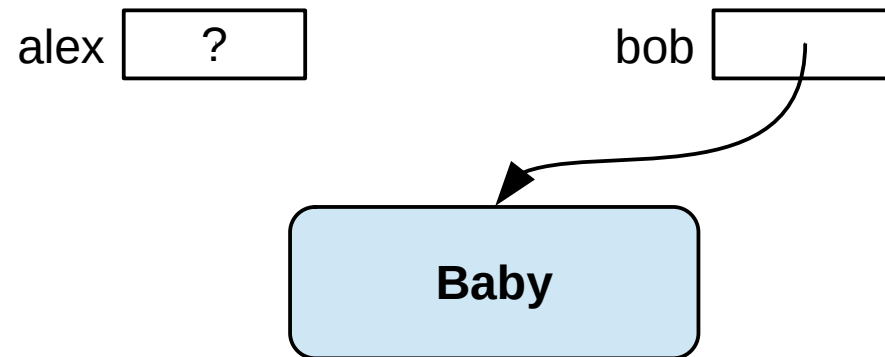
pass by value:
value is copied

i	5
j	10
PI	3.14

Basics of Java Programming

- **Memory management**
 - Working with objects: be careful

```
Baby alex, bob;  
bob = new Baby(...);
```

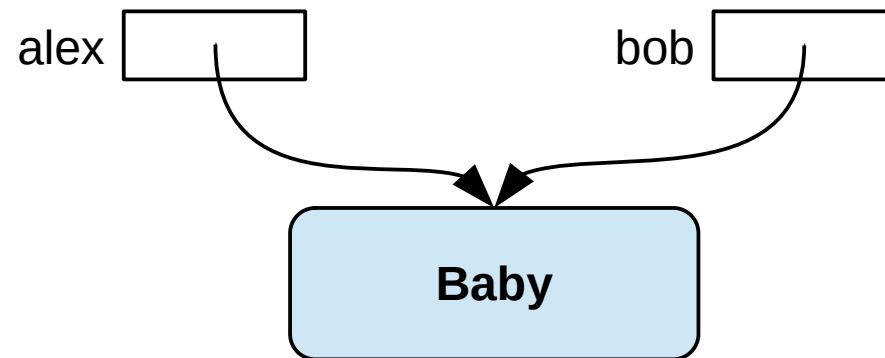


Basics of Java Programming

- **Memory management**
 - Working with objects: be careful

```
Baby alex, bob;  
bob = new Baby(...);  
alex = bob;
```

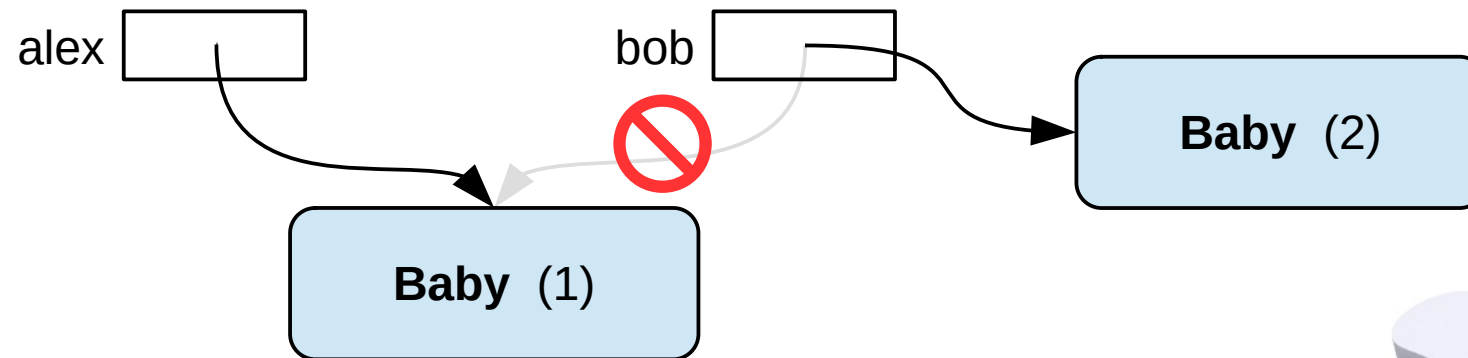
pass by reference:
only reference is copied,
not the entire object



Basics of Java Programming

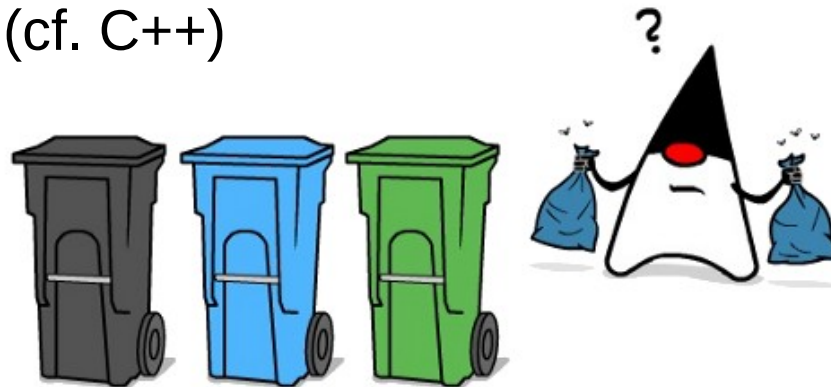
- **Memory management**
 - Working with objects: be careful

```
Baby alex, bob;  
bob = new Baby(...); // (1)  
alex = bob;           // (1)  
bob = new Baby(...); // (2)
```



Basics of Java Programming

- **Memory management**
 - Working with primitives: pass by value
 - Working with objects: pass by reference
- **Lifetime of objects**
 - Garbage collector: automatic release of memory after use
 - No memory leaks (cf. C++)



Basics of Java Programming

- **Scoping: visibility and lifetime of variables**

- Indicated by curly brackets
- Primitives

```
int x = 10;
```

```
{
```

```
    int y = 15;
```

```
}
```

```
...
```

both x and y available

only x available

- Objects

- Same behavior for object reference (in the stack)
- Object itself survives the scope (in the heap)

Basics of Java Programming

- **Package: the library unit**
 - A package is a collection of class files
 - A mechanism to manage “namespaces” and to avoid clashes with names
 - Loading a package with the **import** keyword
 - Adding a class to a package with the **package** keyword
 - File must belong to the directory specified by package structure

```
package mypackage;  
  
public class MyClass {  
    ...  
}
```

```
import mypackage.*;  
  
...  
MyClass m = new MyClass();  
...
```

Basics of Java Programming

- **Access modifiers**
 - Purpose: enforcing rules to work with classes/objects
 - Protection of data / methods for internal use
 - separation between interface and implementation
 - Prevention of abuse
 - keep integrity of objects
 - Keywords: **public** – **private** – **protected**
 - Public: visible to the world (everybody outside and inside the class)
 - Private: visible only to the class
 - Protected: visible to the package and all subclasses (inheritance)
 - Default (friendly), no keyword: visible to the package



Basics of Java Programming

- **Baby example**
 - Let's update the baby class with access control

```
public class Baby {  
    private String name = "Unknown";  
    private boolean isMale = true;  
    private double weight = 0.0;  
    private int nbPoops = 0;  
    private static int nbBabiesMade = 0;  
  
    public Baby(String n, boolean m, double w) { ... }  
    public void sayHi() { ... }  
    public void eat(double food) { ... }  
    public void poop() { ... }  
}
```


Basics of Java Programming

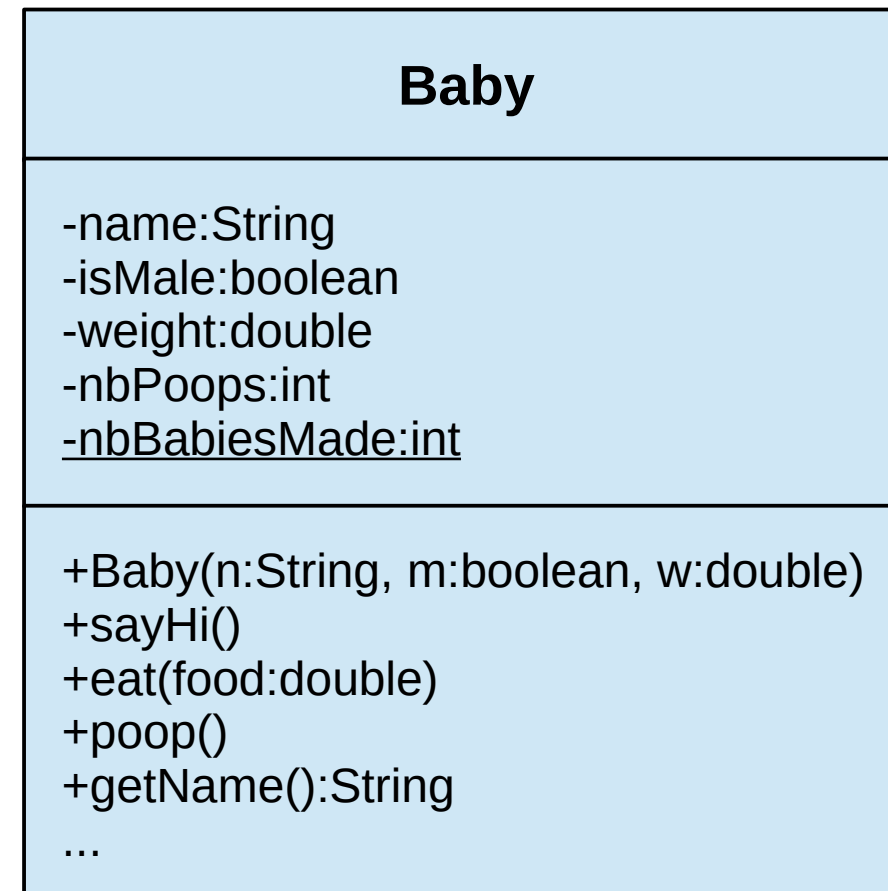
- **Baby example**
 - Let's update the baby class with access control

```
public class Baby {  
    private String name = "Unknown";  
    ...  
    private static int nbBabiesMade = 0;  
  
    ...  
    public String getName() { return name; }  
    public double getWeight() { return weight; }  
    public int getNbPoops() { return nbPoops; }  
    ...  
    public static int getNbBabies() { ... }  
}
```


Basics of Java Programming

- **UML class diagram**

- Each class is represented by a box
 - Sections: name, variables, methods
 - Special codes for modifiers:
 - public (+), private (–), protected (#)
 - static (underlined)
- Relationships between classes
- Keep it simple
 - Complete diagram is heavy
 - Display only the info required for your purpose



Basics of Java Programming

- **Correct use of names**

- Rules:

- Sequence of Unicode letters and digits, dollar sign “\$”, underscore “_”
 - Beginning with a letter and case-sensitive

- Naming conventions:

- Class names are a collection of nouns, with the first letter of each word capitalized
 - Variable names (object references, arguments, ...) have the first letter lowercase, and first letter of other words capitalized
 - Method names are verbs with the first letter lowercase, and first letter of other words capitalized
 - Constants are all uppercase, words separated by underscores
 - Package names are all lowercase



Basics of Java Programming

- Making a (simple) Java program

- Objective

- A program that can manage bank accounts
 - E.g., changing balance by deposits and withdrawals, computing interests, ...

- Step 1: what do we need ?

- A class BankAccount
 - Each individual account = object
 - Keep track of balance
 - Make deposits and withdrawals
 - Compute interest
 - ...
 - ...



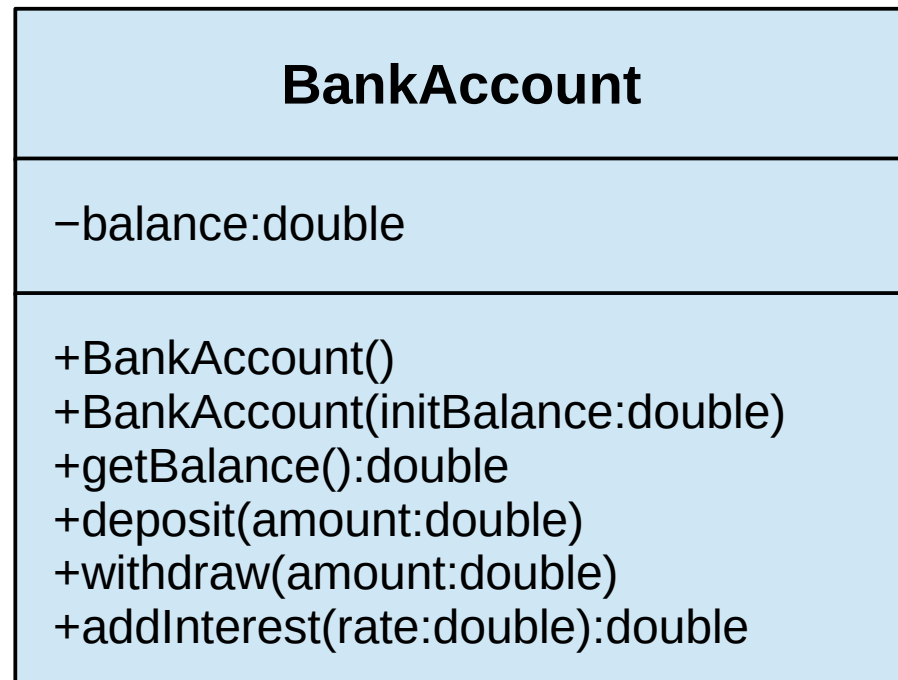
Basics of Java Programming

- Making a (simple) Java program
 - Step 2.1: defining interfaces

```
BankAccount myAccount = new BankAccount();  
  
double amount1 = 100, amount2 = 50;  
myAccount.deposit(amount1);  
myAccount.withdraw(amount2);  
  
double rate = 0.01;  
double interest = myAccount.addInterest(rate);  
  
double balance = myAccount.getBalance();  
...
```

Basics of Java Programming

- Making a (simple) Java program
 - Step 2.2: UML class diagram



UML diagrams help you understand, discuss, and design software programs

Basics of Java Programming

- Making a (simple) Java program
 - Step 3.1: internal data (class variables) + get/set methods

```
public class BankAccount {  
    private double balance; // balance (EUR)  
  
    /**  
     * Gets the current balance of the bank account.  
     * @return the current balance  
     */  
    public double getBalance() {  
        return balance;  
    }  
    ...  
}
```

Javadoc standard
(skipped later on)

Basics of Java Programming

- Making a (simple) Java program
 - Step 3.2: constructors

```
public class BankAccount {  
    private double balance; // balance (EUR)  
  
    ...  
    public BankAccount() {  
        balance = 0.0;  
    }  
  
    public BankAccount(double initBalance) {  
        balance = initBalance;  
    }  
    ...  
}
```

constructor overloading:
unique set of parameters

Basics of Java Programming

- Making a (simple) Java program
 - Step 3.3: other methods

```
public class BankAccount {  
    private double balance; // balance (EUR)  
  
    ...  
    public void deposit(double amount) {  
        balance = balance + amount;  
    }  
  
    public void withdraw(double amount) {  
        balance = balance - amount;  
    }  
    ...  
}
```


Basics of Java Programming

- Making a (simple) Java program
 - Step 3.3: other methods

```
public class BankAccount {  
    ...  
    /**  
     * Adds interest to the bank account.  
     * @param rate - the interest rate  
     * @return the computed interest  
     */  
    public double addInterest(double rate) {  
        double interest = balance * rate;  
        balance = balance + interest;  
        return interest;  
    }  
}
```

Basics of Java Programming

- Making a (simple) Java program
 - Step 4: the main program

```
import java.util.*;

public class BankProgram {
    public static void main(String[] args) {
        BankAccount account = new BankAccount();
        Scanner reader = new Scanner(System.in);
        System.out.print("Deposit in Euro: ");
        account.deposit(reader.nextDouble());
        reader.close();
        System.out.println("Account Balance: "
            + account.getBalance() + "EUR");
    }
}
```



Basics of Java Programming

- **Methods: visibility of variables**

```
public double addInterest(double rate) {  
    double interest = balance * rate;  
    balance = balance + interest;  
    return interest;  
}
```

class variable:
balance

parameter:
rate

local variable:
interest

- Variables inside scope of method
- Available during method execution

Basics of Java Programming

- Methods: interchanging data

```
public double addInterest(double rate) {  
    double interest = balance * rate;  
    balance = balance + interest;  
    return interest;  
}
```

primitives: pass by value
objects: pass by reference

```
interest = myAccount.addInterest(rate);
```



Basics of Java Programming

- **Methods: overloading**

- We can deduce meaning from the context

“wash the shirt”

“wash the car”

“wash the dog”

instead of

“washShirt the shirt”

“washCar the car”

“washDog the dog”

- Methods can have the same name, but unique set of parameter types

```
public void withdraw(double amount) {  
    ...  
}  
public void withdraw(double amount, double fee) {  
    ...  
}
```

Basics of Java Programming

- Ready for an exercise...

