5COSC001W – Object Oriented Programming
Week 1

Dr. Barbara Villarini

b.villarini@westminster.ac.uk

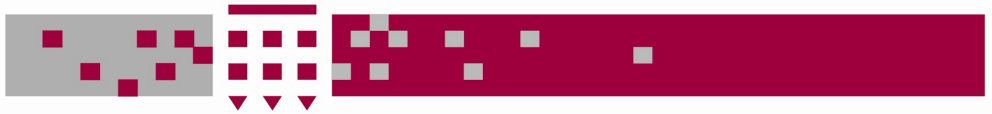
5COSC001W - OOP

This module :

- introduces the concepts of objects and classes
- teaches to apply principles of object oriented programming, OOP analysis and design in tackling programming problems
- aims to extend the programming skills you acquired in the first year

Module contents

- Object Oriented Principle and characteristic
- Classes and Objects. The usage of APIs
- Inheritance, polymorphism, abstraction, encapsulation.
- How to design class. Introduction to UML
- Computational modelling
- Graphical User Interfaces
- Concurrency and multithreading
- Defensive programming
- Input/Output and streams
- Fundamental design patterns
- Testing, unit testing



Objectives

After this course you should be able to:

- **Tackle** more complicated programming problems
- **Decompose** programming problems using object oriented analysis and design
- Use **most** of the object-oriented programming concepts when writing programs
- **Improve** the knowledge of Java programming language

Learning outcome

- Identify and justify good practices in development of OO software
- Apply acquired knowledge of concepts and programming languages based on OO principles
- Design, implement and test application based on OOP
- Implement GUI interfaces using OOP languages
- Describe, justify apply concurrency principle and how threads can be implemented in an OOP language

Assessments

Consists of :

- **Coursework** – Classes, inheritance, polymorphism, OOP, GUI
- **In class test** – All topics in the module

Coursework deadline:

- Deadline Coursework: 4 January at 13:00 2023

In class test date:

- In-class test during your tutorial (seminar) slot in week 12

UNIVERSITY OF WESTMINSTER

Plagiarism and Academic Misconduct

"In other words, plagiarism is an act of fraud. It involves both stealing someone else's work and lying about it afterward."

(http://www.plagiarism.org/plagiarism-101/what-is-plagiarism)

- **Do not copy the code from other students**
- **Do not copy the code from external source**
- **You will be asked to demonstrate the understanding of your work**

https://www.westminster.ac.uk/study/current-students/resources/academic-regulations/academic-misconduct

Books and resources

- ***Big Java***, Horstmann, C. 4th edition. Wiley.

Further reading:

- ***Java. A Beginner's Guide***, Herbert Shildt

- ***The object Oriented Thought Process***, 4th Edition, M. Weisfield, Addison Wesley
- ***Design Patterns: elements of reusable object-oriented software***, Gemma, Helm, Johnson, Vlissides. Addison Wesley

My expectations of you

- You attend **all** lectures and tutorials

– Lectures are online and the pdf will be available on Blackboard before lecture

– Tutorials are on-site – look at the timetable for your seminar slot

- Attend all lectures and tutorials **on time**
- You do all tutorial exercises and in-lecture exercises
- Do not do your coursework during tutorials, there will be specific session for that
- Submit coursework on time
- Flag any problems you have early

Motivation: Why programming is important?

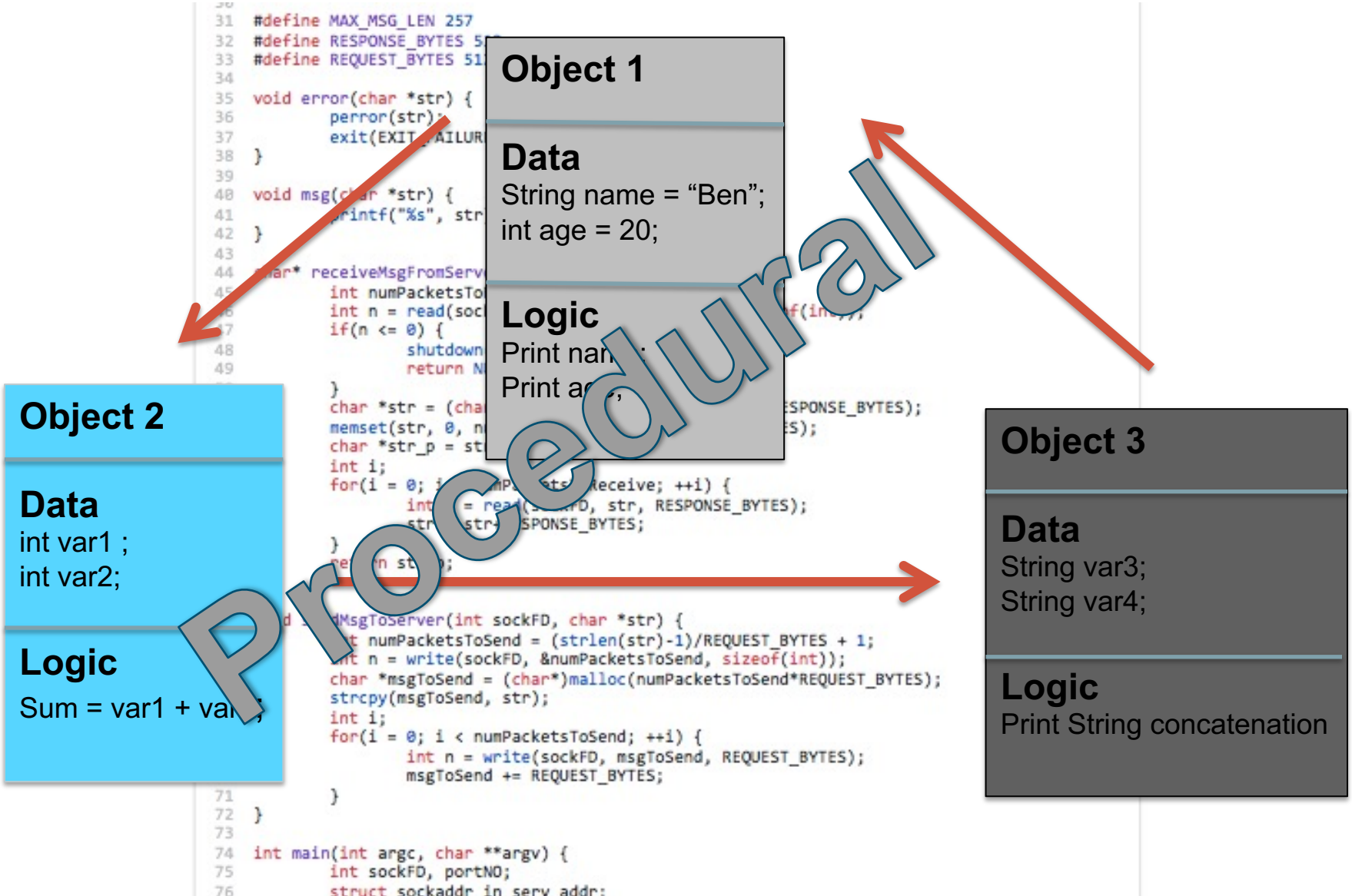
- Even if you will not become a professional programmer, learning programming is a valuable life skill. It involves:

➢ creativity: programming involves problem solving skills...

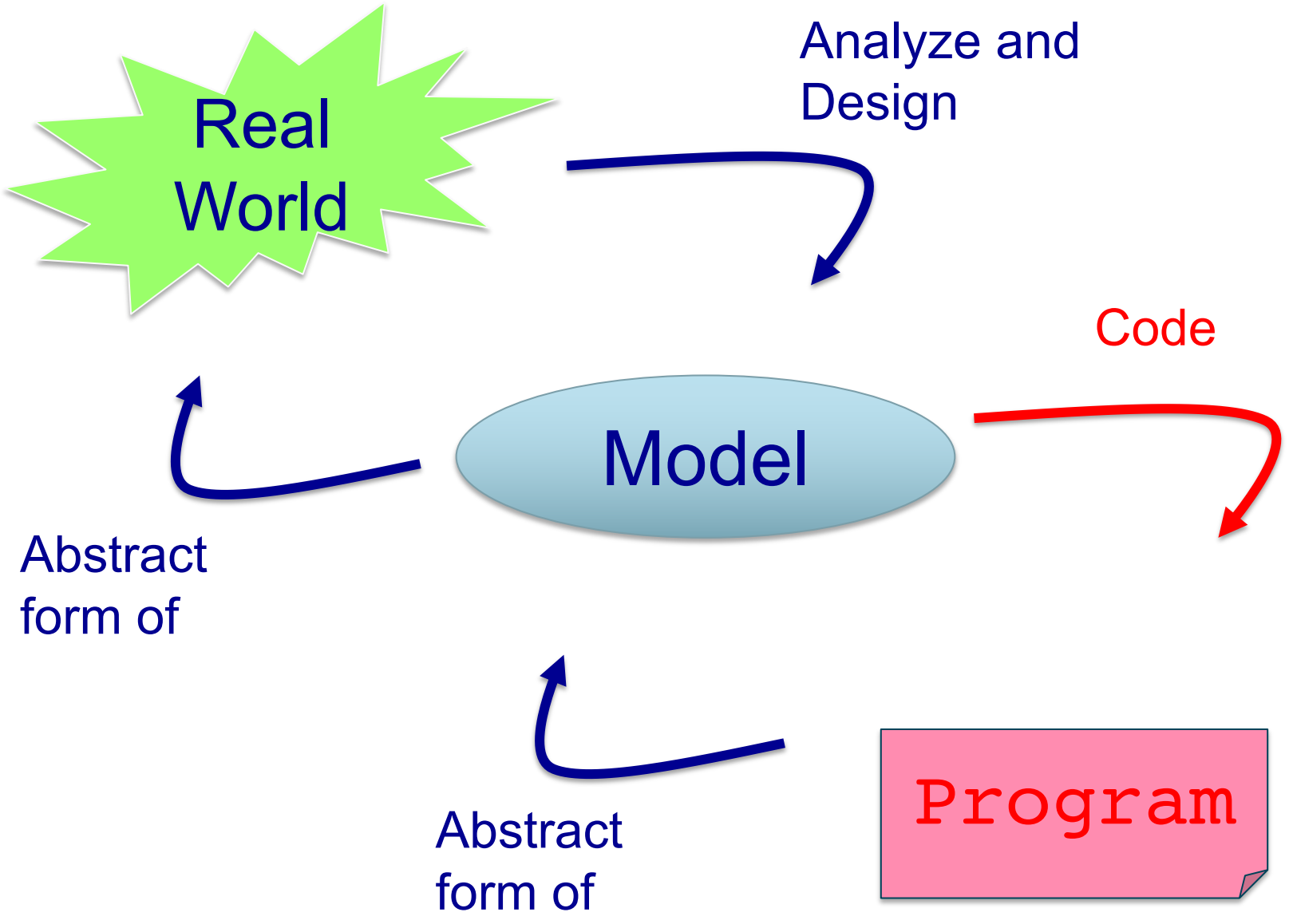
➢ systematic thinking: programming teaches you to reason logically and consequently…

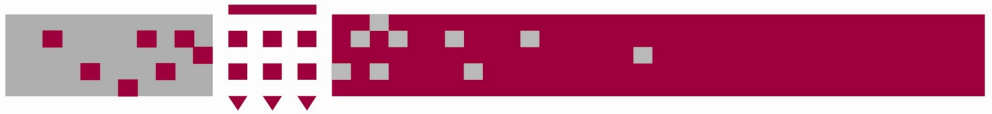
➢ collaborative work: coding within a team...

Object Orientation Programming



Model


Real World
Analyze and Design
Code
Model
Abstract form of
Abstract form of
Program

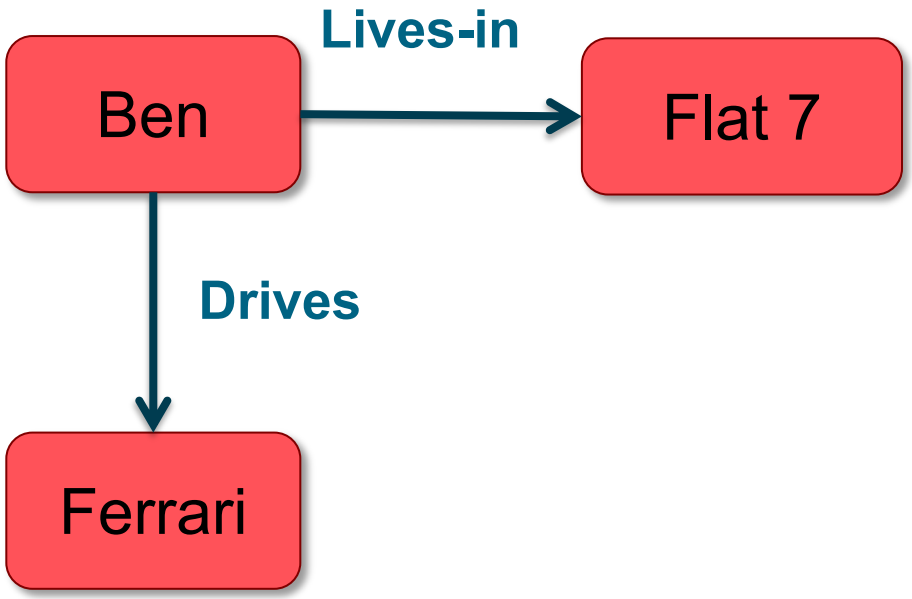


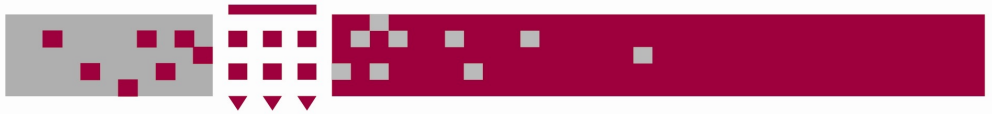
So what is “Object Oriented” about?

- A program typically creates a **model** of a part of the “real” world
- The parts of the model are the **objects** that can be identified in the problem, and these will be included in the software model
- Objects can be categorised into **classes**
- all objects that share similar characteristics or behaviors, that are of the same kind, belong to the same class
- an object that belongs to a class is an **instance** of this class

Example – OO Model

- Objects
  - Ben
  - Flat 7
  - Ferrari
- Interactions
  - Ben lives in the house
  - Ben drives the car





OO - Advantages

- People think in terms of objects
- OO models map to reality
- Therefore, OO models are
  - easy to develop
  - easy to understand

Objects

An object is

- Something tangible (person, pen, mug)
- Something that can be apprehended intellectually (Time, Date)
- An object has **state**, **behavior**, **identity**

Person Object	**Person Object**	**Bank Account Object**
Name : “Bob” Age: 22	Name : “Jane” Age: 33	number: 01 23 45 balance: 200 £
speak() walk()	speak() walk()	deposit() withdraw()

Classes

Classes

Type

- Name: What is it?
  - Person, BankAccount, Employee, Time, etc.

Properties, data -> instance variables

- Attributes: What does it describe?
  - Name, Age, Balance, salary, etc.

Operations -> instance methods

- Behavior: What can it do?
  - Speak, deposit, work, etc.

In summary, What are Objects and Classes?

An object is

- Something tangible (Ben, Ferrari)
- Something that can be apprehended intellectually (Time, Date)
- An object is an **Instance of a class.**

  - Ben is an instance of the class Persons: a **specific object** that belongs to the class Persons

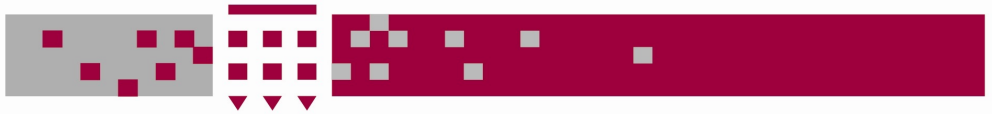
- We want our **class** to be a grouping of conceptually-related state and behaviour

Example – Ben is a Tangible Object instance of the class Person:

- State (attributes)
  - Name
  - Age
- Behavior (operations)
  - Walks
  - Eats
- Identity
  - His name

Example – 11:00:00 is a time and is an Object Apprehended Intellectually, it is an instance of the class Time:

- State (attributes)
  - Hours
  - Minutes
  - Seconds
- Behavior (operations)
  - Set Hours
  - Set Minutes
  - Set Seconds
- Identity
  - Would have a unique ID in the model

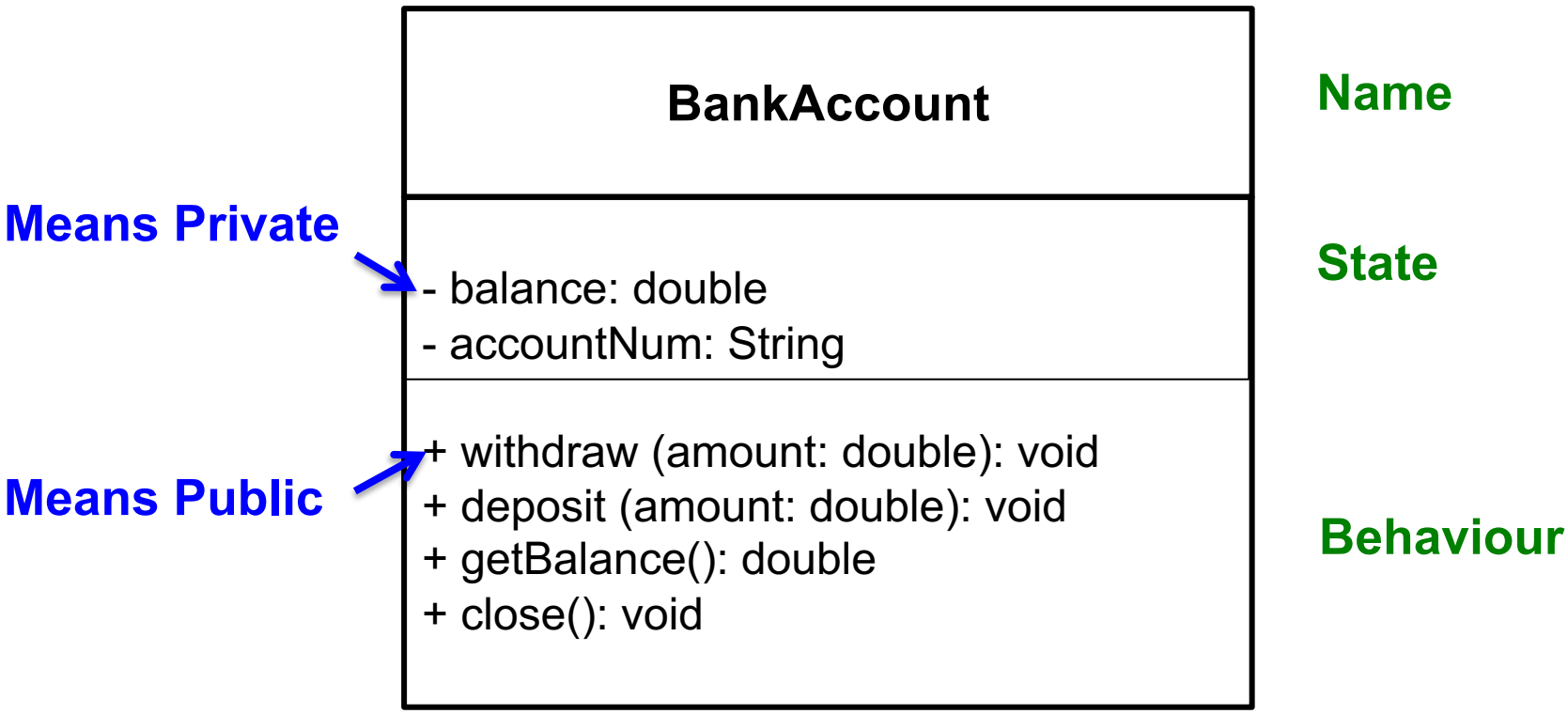


Our first class

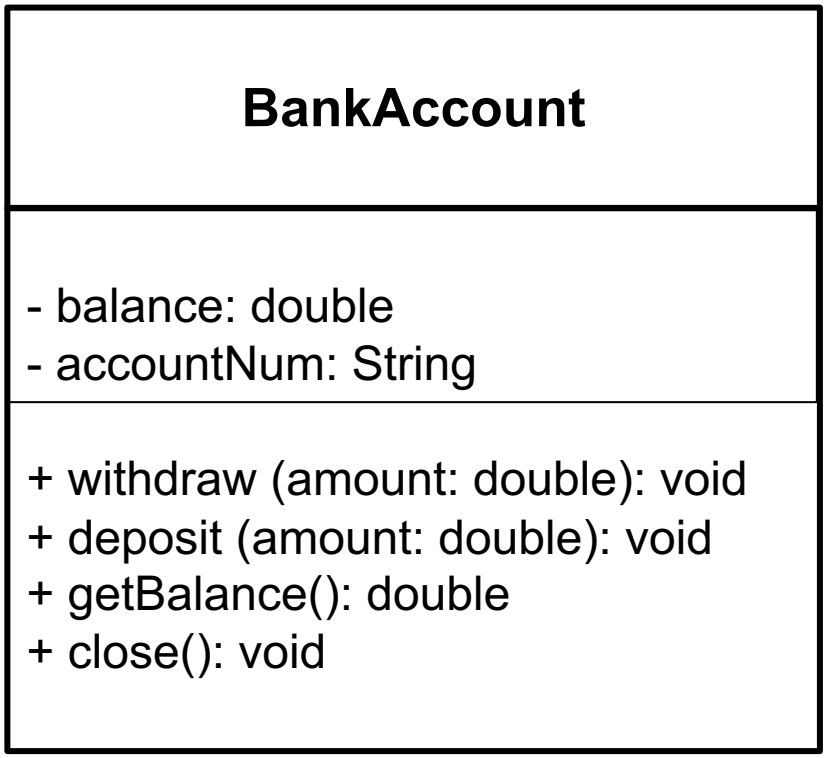
- Suppose you write code for a bank, and you are asked to write a class Account

- What is state, behavior for a bank Account?

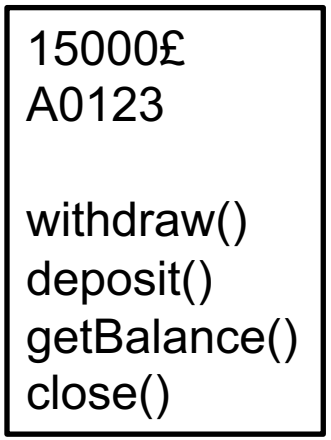
Representing a Class Graphically (UML)



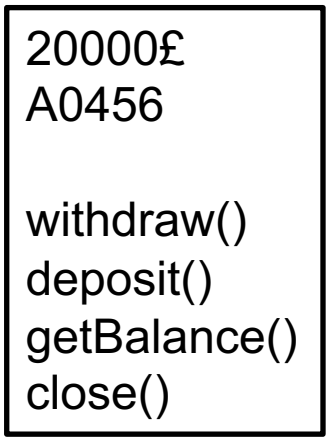
Example: BankAccount class



Class



BobAcc



JaneAcc

Object (instance)

The Account Class – Declaration


```
public class Account
{       private double balance;
        private String accountNum;

        public Account(double initialBalance, String accNum)
        { some code }
        public Account()
        { some code }

        public void withdraw(double amount)
        { some code }
        public void deposit(double amount)
        { some code }
        public double getBalance()
        { some code }
        public void close()
        { some code }
}
```

Name of the Class

State: instance variables

Constructors: to initialize objects, to create instances

Behavior: instance Methods, to change or show the state of Account

Class Declaration

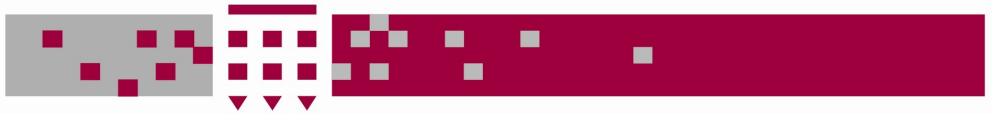
- In general:
- By convention a class name always starts with a capital letter.

➢ A class declaration contains declarations of **instance methods** and **instance variables**

➢ Typically, a class declaration also contains declarations of **constructors**: their job is to initialise objects

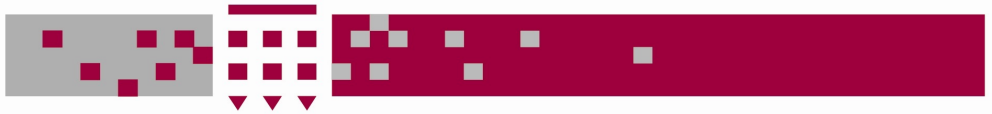
```
public class class_name {
        variable-declarations
        constructor-declarations
        method-declarations
}
```

➢ The order is not important, but this is a standard way of class declarations



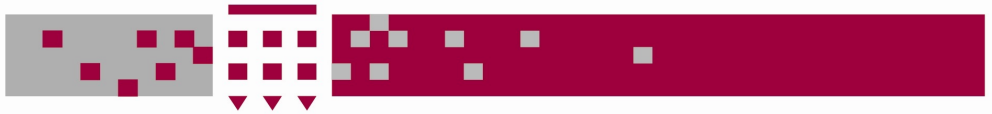
Instance Variables

- An instance variable can hold a single value
  - e.g. `private double balance`;
- But also, an instance variable can hold an object
  - let us not worry about the details of this for now
- Instance variables are typically defined as private
  - this facilitates **information hiding** between classes
  - more on access modifiers (e.g. private) later
  - **information hiding essential in industry**



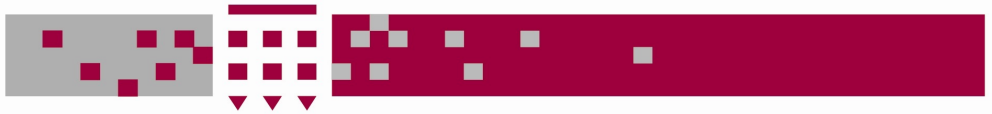
Instance Methods

- Instance methods represent the behavior and alter the state of an object, they manipulate an object
- In the Account example:
  - ***deposit***: increases the balance of the account by a specified amount
  - ***withdraw***: decreases the balance of the account by a specified amount
  - ***getBalance***: prints the balance of the account
  - ***close***: closes the account



The Account Class

```
public class Account
{
private double balance;
...
public void withdraw(double amount) {
balance -=amount;
}
public void deposit(double amount) {
balance += amount;
}
public double getBalance() {
return balance;
}
public void close() {
 balance = 0;
}
}
```



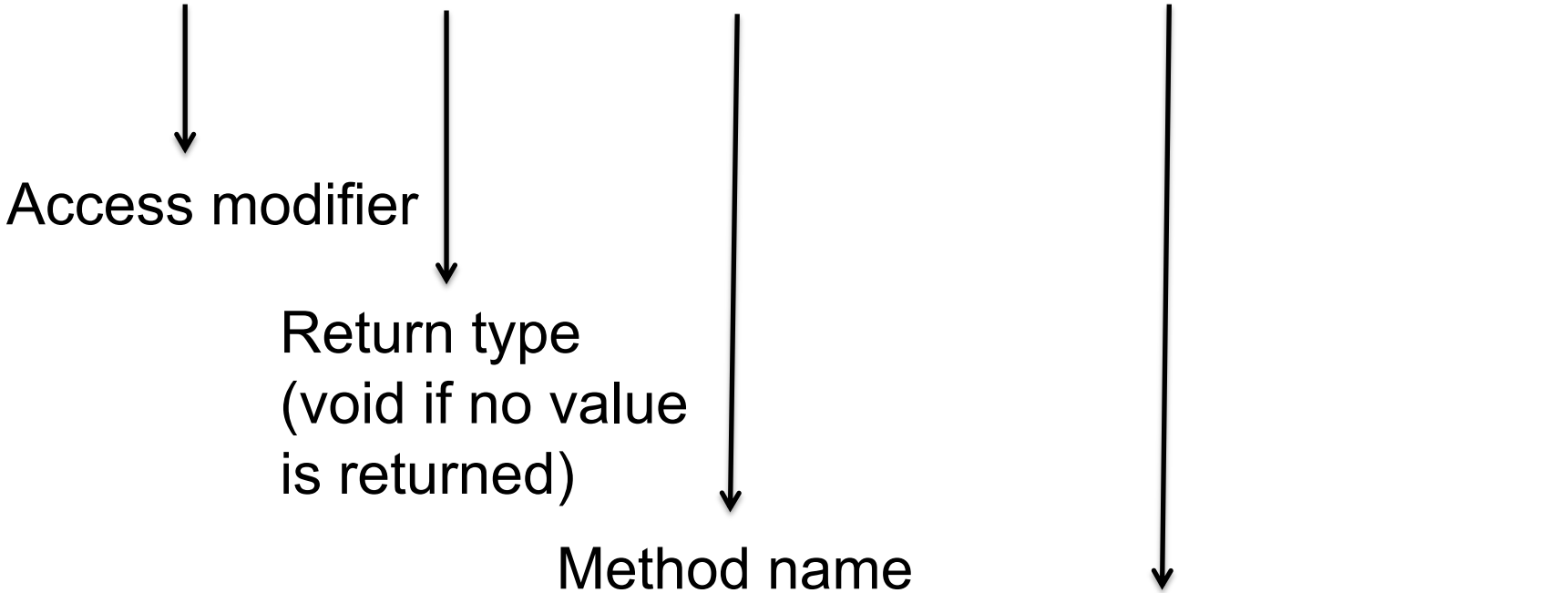
Instance Methods

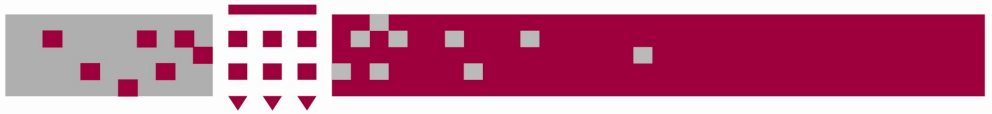
- Instance methods also provide the interface of classes
  - ➢ different objects can communicate with each other via their instance methods

- To achieve this, instance methods are typically defined as **`public`**
  - ➢ can be accessed by members of other classes

Declaring instance methods

```
public void deposit(double amount)
```





Access Modifier

- The declaration of an instance variable, a constructor, an instance method (even of a class) begins with an access modifier (public or private)
  - **Public**: the entity can be accessed by other classes
  - **Private**: the entity is only accessible from within the class itself

- In general (there are exceptions) instance variables will be `private`, and instance methods and constructors will be `public`

OOP Principles

Abstraction

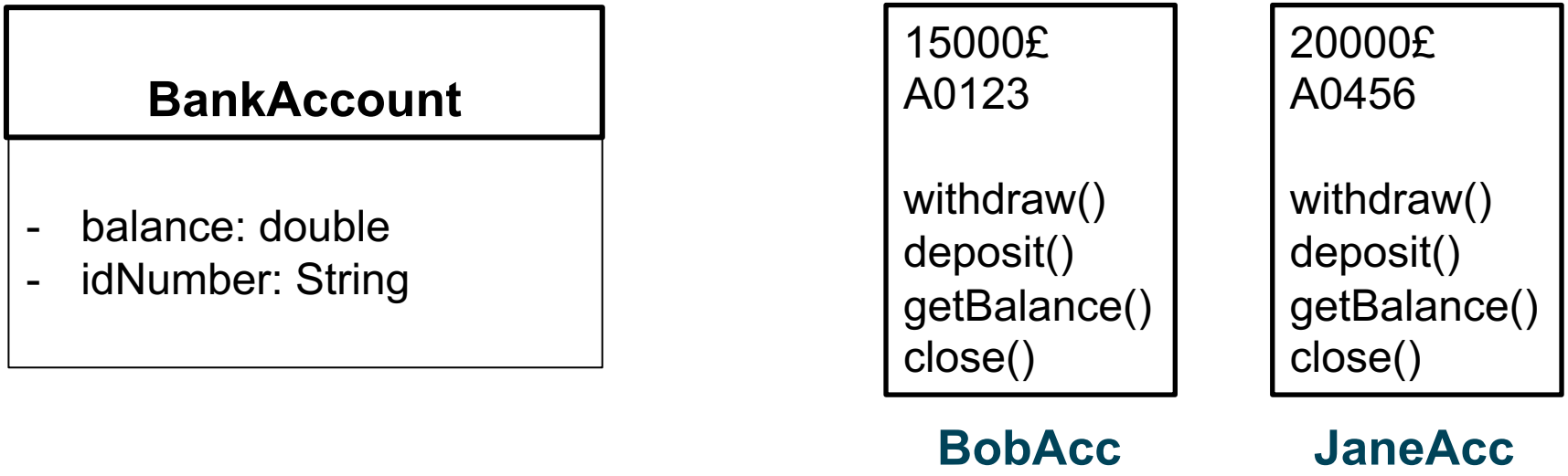
Polymorphism

Inheritance

Encapsulation

Abstraction

- Focus on the essential quality of an object
- Discard what is irrelevant
- We create one Class to represent several objects



Encapsulation

- wrapping the data (variables) and code acting on the data (methods) together as a single unit.
- the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.

Black Boxing

How to do it?

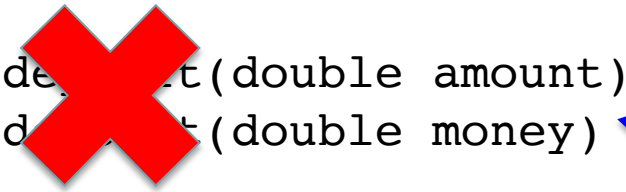
- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

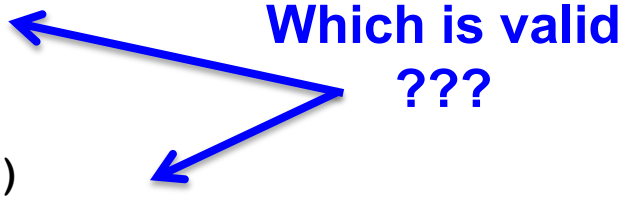
Method Overloading

- Instance methods can be overloaded, i.e. more than one method can have the same name inside the same class
- For this to work, overloaded methods must have different number of parameters, or parameters of different types

```
1) public void de   t(double amount)
   public void d    t(double money)

2) public void deposit(double amount)
   public void deposit(double amount, double interest)
```

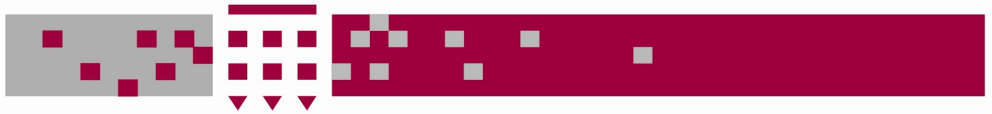




Constructors

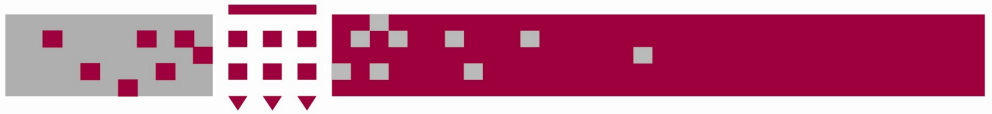
- When an object is created, its instance variables are initialised by a **constructor**

```
public class Account
{
        private double balance;
        public Account(double initialBalance)
        { balance = initialbalance;}
        public Account()
        { balance = 0;}
…
}
```



Constructors

- The constructor **MUST** have the same name as the class name
- A class can have more than one constructor
  - like the example in the previous slides
  - constructor overloading
  - same rules apply as for method overloading

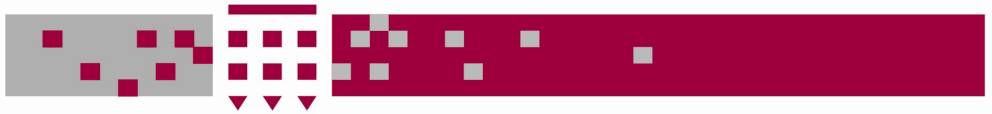


How do we create an object in Java?

- We need to call the constructor of the class in order to create instances of the class
- For this we need the **new** keyword

```
Account account1 = new Account(100);
```

- the above line declares a variable named *account1* of type Account
- it gives to the variable account1 the value *new Account(100)*
- similar to int x =5 where we declare a variable named x of type int and give to the variable x the value 5



Type Account

- *Account* is actually a *new type*.
- This allows *Account* to be used for declarations such as:

```
Account account1= new Account(100);
```

- In fact, Account is a *User Defined Type*.

Call constructors

- We need to call the constructor of the class in order to create instances of the class
- For this we need the new keyword

```
account1 = new Account();
account2 = new Account(100);
```

  - we need to call the constructor with the correct parameters and types
  - remember, we used 2 constructors for the Account class in our previous examples

Calling instance method

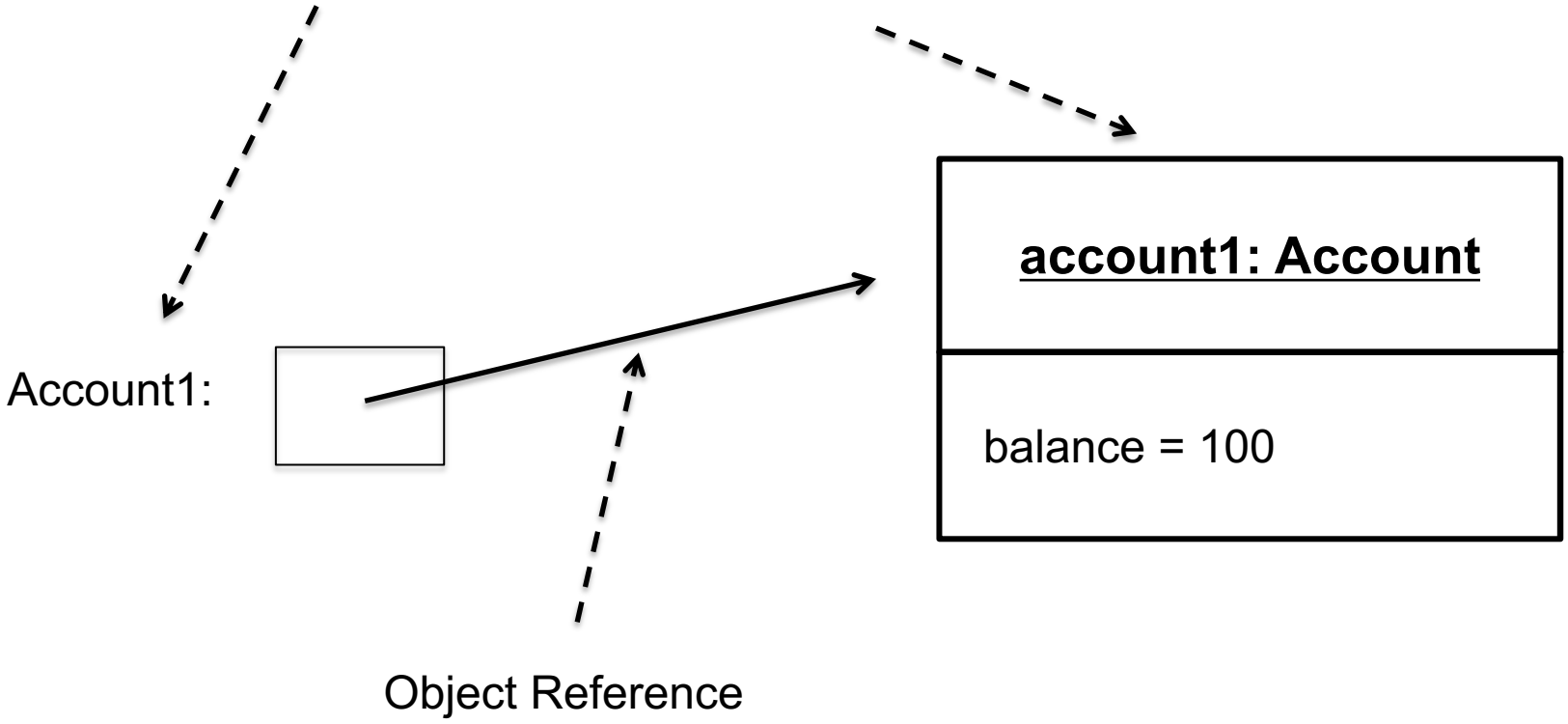
- Once an object has been created, operations can be performed on it by calling the instance methods of the object's class

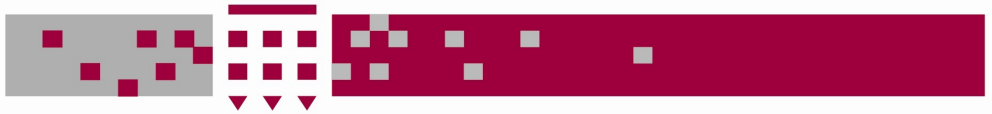
`object_name.method_name(arguments)`

```
account1.deposit(1000);
account2.withdraw(250);
account2.close();
```

Object References

```
Account account1 = new Account(100);
```





Reference

- A variable of a class type holds a *reference* to an object.

– A reference is a pointer (form of memory address).

- Variable doesn't hold the object itself.
- The variable can go out of scope but the object can *still* exist (providing it is referenced by some other variable).
- One object can be referenced by several references and, hence, variables.

Copying Variables of Primitive Data Types and Object Types

Primitive Type Assignment
i = j

Before

i: 1

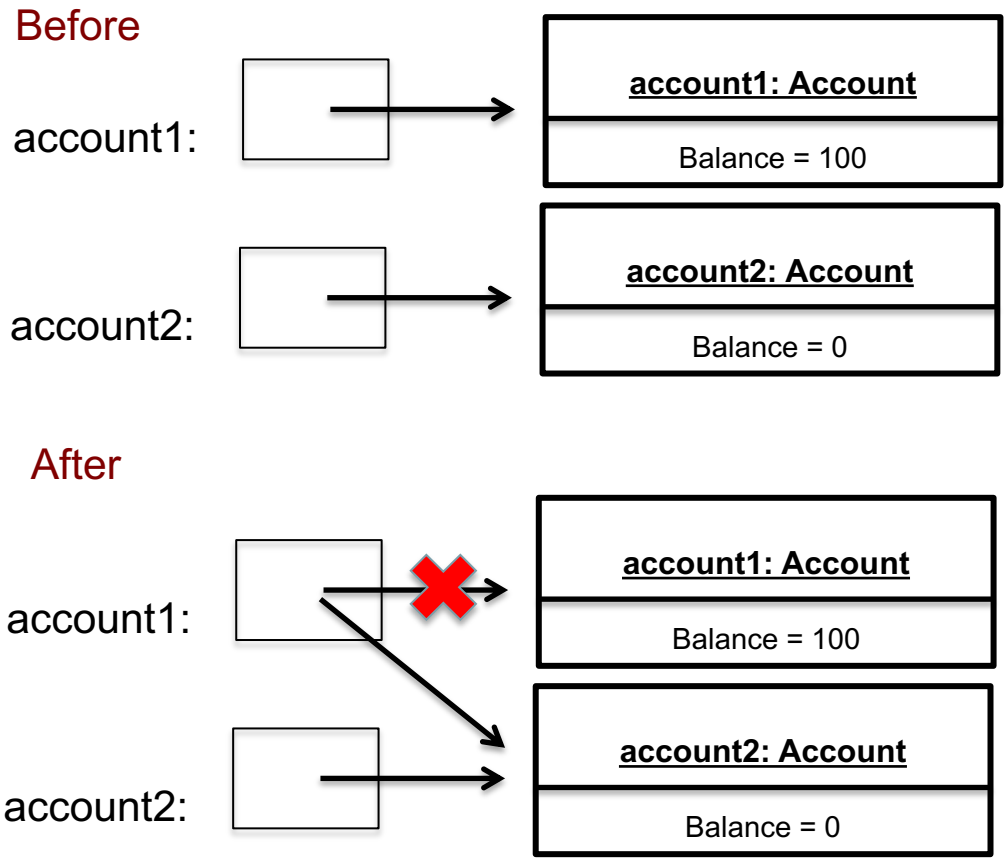
j: 2

After

i: 2

j: 2

Object type Assignment
account1 = account2

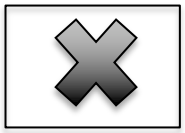


Garbage Collection

- As shown in the previous figure, after the assignment statement `account1 = account2`, `account1` points to the same object referenced by `account2`. The object previously referenced by `account1` is no longer referenced. This object is known as garbage.
- Garbage is automatically collected by JVM.

Null Reference

- `null` keyword.
- No object is referenced, so no methods can be called.
- **`Account account1= null;`**
- Default value if variable not initialised.

account1: 

Object reference Parameters

- You can pass an object reference as a parameter to a method:

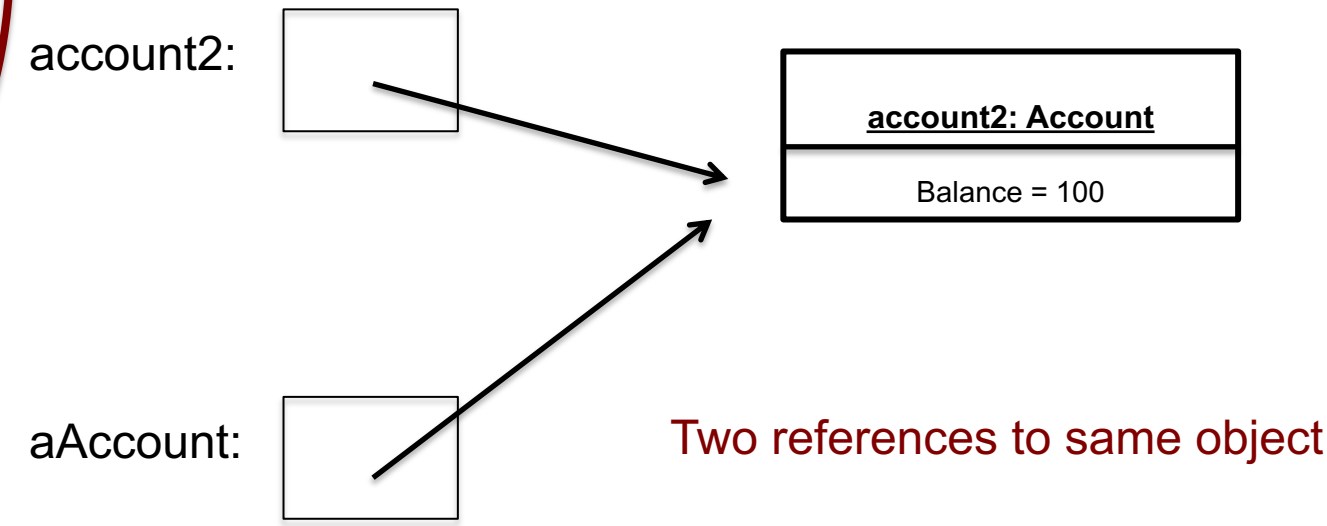
```
void moveAccount(Account account) {
      ... // Use account in method body
}
```

- A parameter variable is declared as normal.

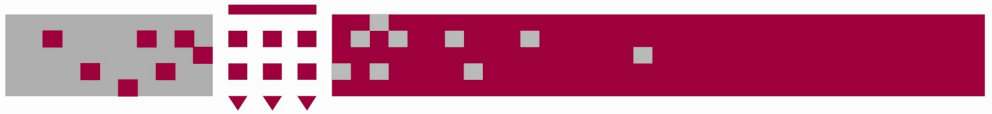
Parameters & References

Method call: **`account1.moveAccount(account2)`**

Calling the instance method defined in the class Account

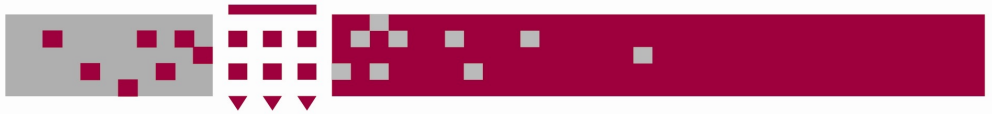



```
void moveAccount(Account aAccount){
    aAccount.deposit(balance);
    balance = 0;
}
```



Object Parameters

- The parameter value is an *object reference*, not an object.
- The parameter variable is initialised to hold a copy of the reference.
- The object is *not copied*.
  – The reference is copied *not* the object.

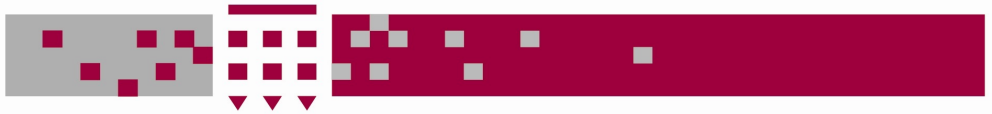


Consequences

- If an object reference is passed as a parameter then:

➢ Changing the object inside the method changes the object outside the method.

➢ They are the same object!



Call-by-value

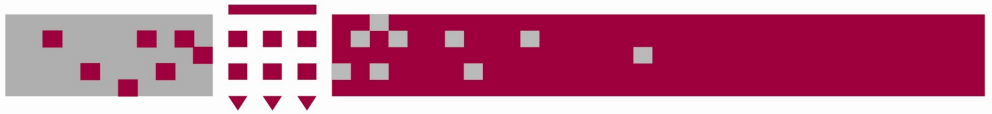
- The parameter passing mechanism used by Java is called “Call-by-value”.
- This means that the value of a parameter is always copied and a parameter variable initialised with the copy.
- Objects are not passed as parameters, only references to objects.
  - The reference is copied.

Return-by-value

- Returning a value from a method works in the same way as parameter passing.

```
public Account findAccount(String name) {
      // find …
      return aAccount;
}
```

- The value returned is a *copy* of the value computed in the return statement.



Summary – What you should know so far…

- Objects and Classes
- Class declaration
- Object as instance of a class
- Constructors
- Object Assignment
- Overloading methods
- Values and References