Session 08: Objects and Classes

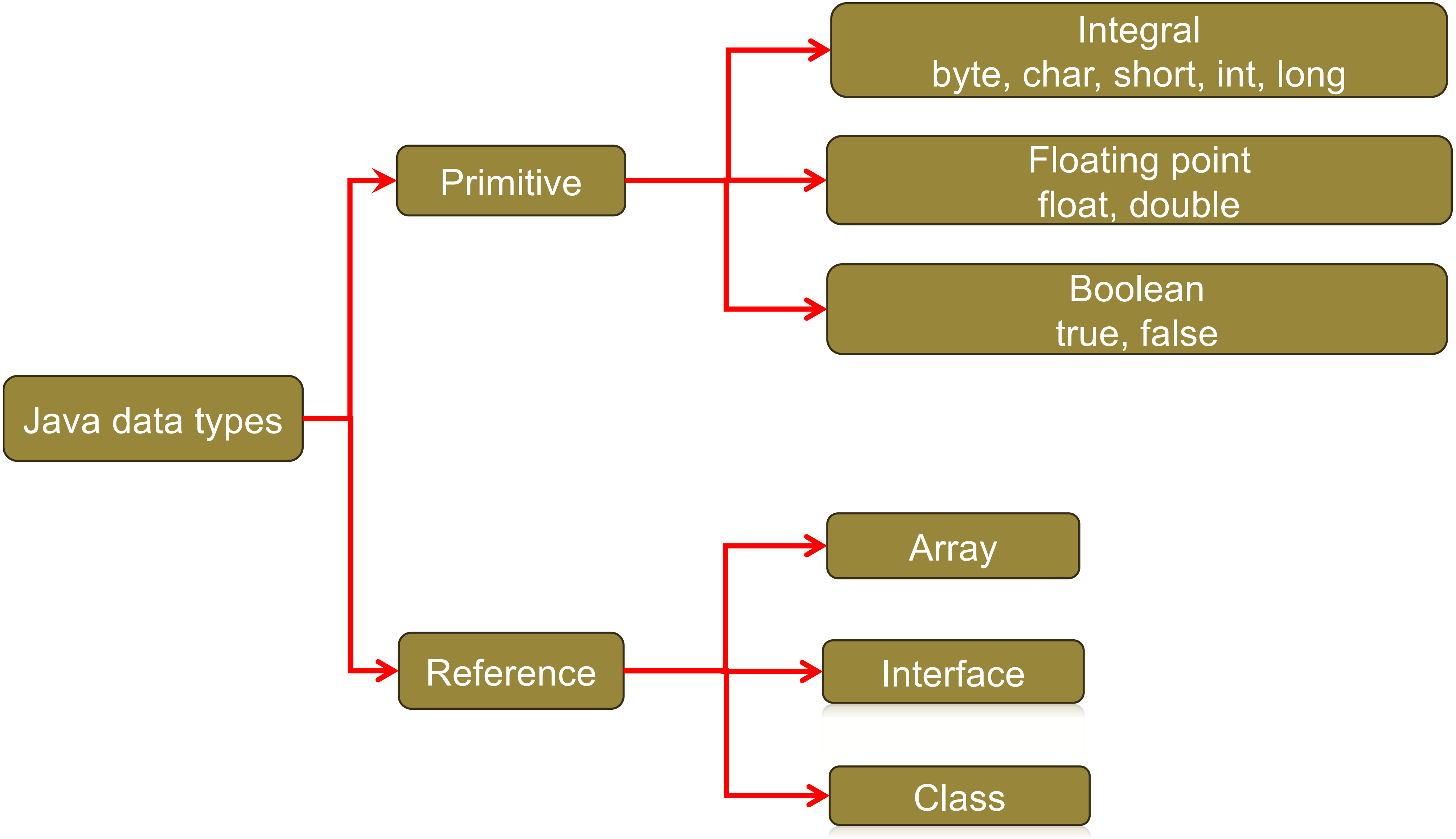
Dr Tomasz Kurowski

t.j.kurowski@cranfield.ac.uk

Session 8 Objects and Classes

- Object Oriented (OO) Programming Concepts
- Creating Objects and Object Reference Variables
  - Differences between primitive data type and object type
  - Automatic garbage collection
- Constructors
- Modifiers (`public`, `private` and `static`)
- Instance and Class Variables and Methods
- Scope of Variables
- Use the **`this`** Keyword
- Case Studies (`Mortgage` class and `Count` class)

Java data types



Java data types

Primitive

Integral
byte, char, short, int, long

Floating point
float, double

Boolean
true, false

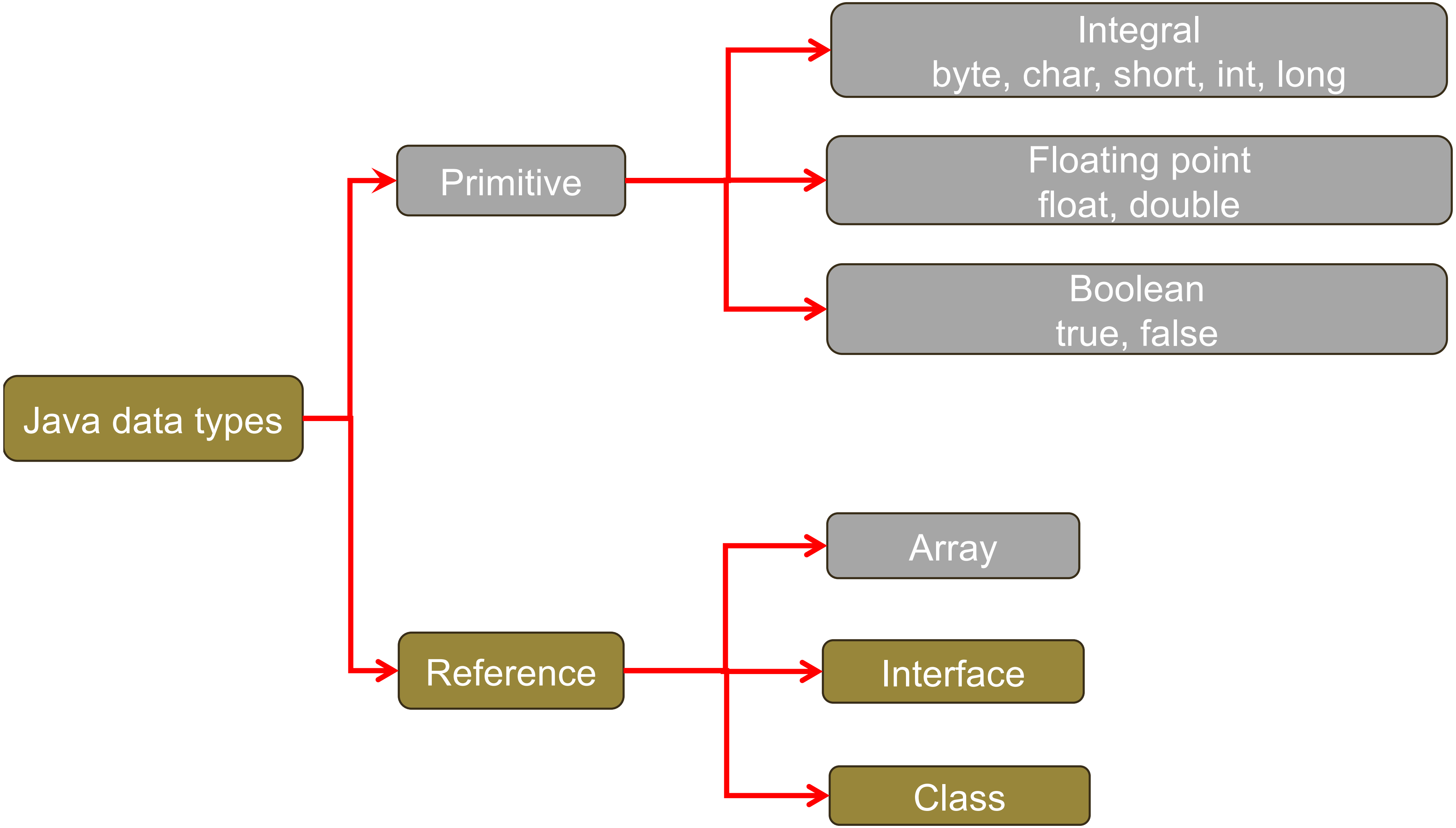
Reference

Array

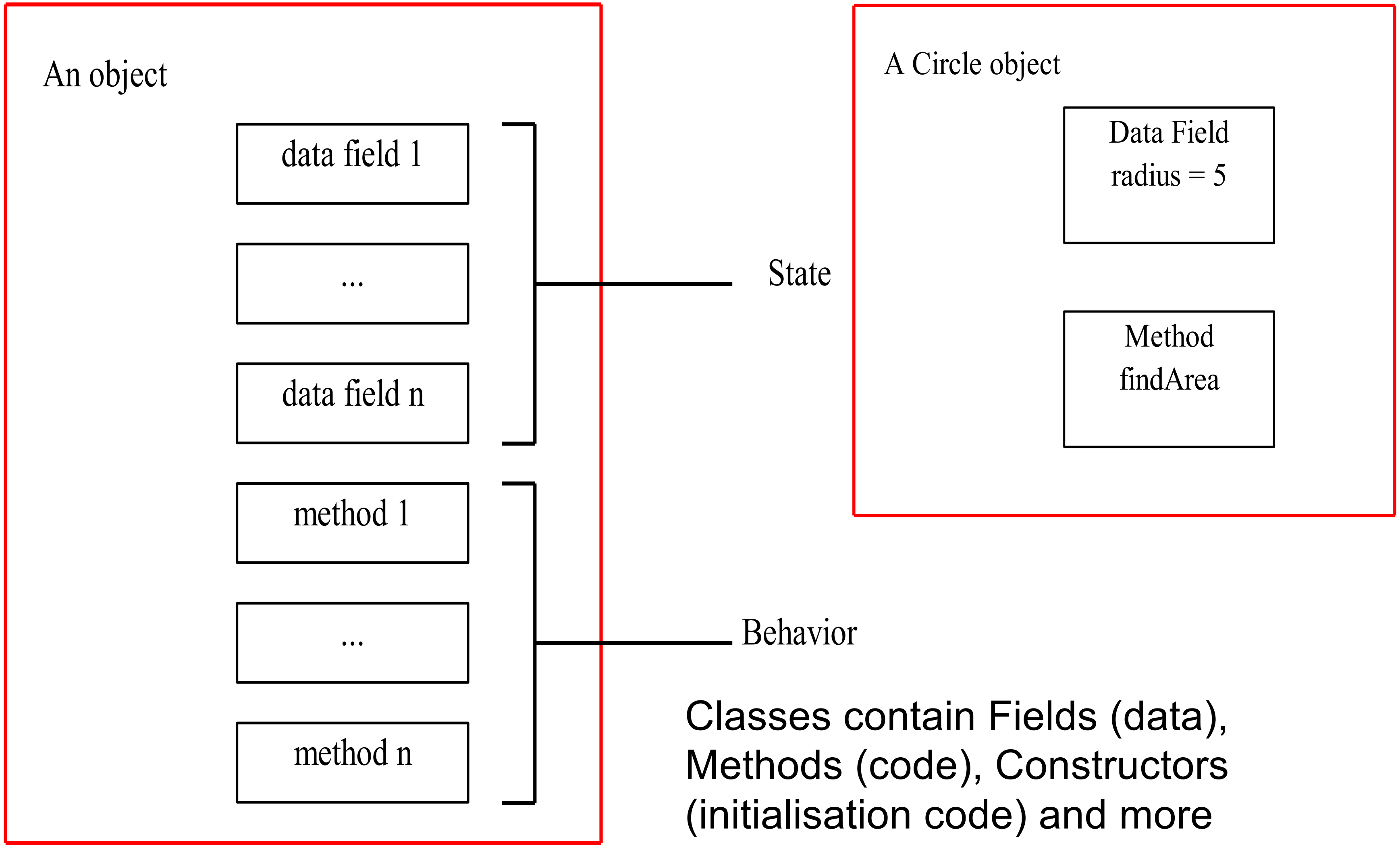
Interface

Class

Java data types



OO Programming Concepts



Classes contain Fields (data), Methods (code), Constructors (initialisation code) and more

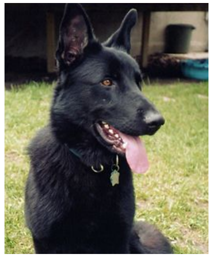
Class Attributes:

- Each object is represented by its attributes (set of values). These attributes can be:
    - Class value: A value associated with a class and every object of the class
    - Instance value: A value associated with a specific object

Class attribute: Example_01

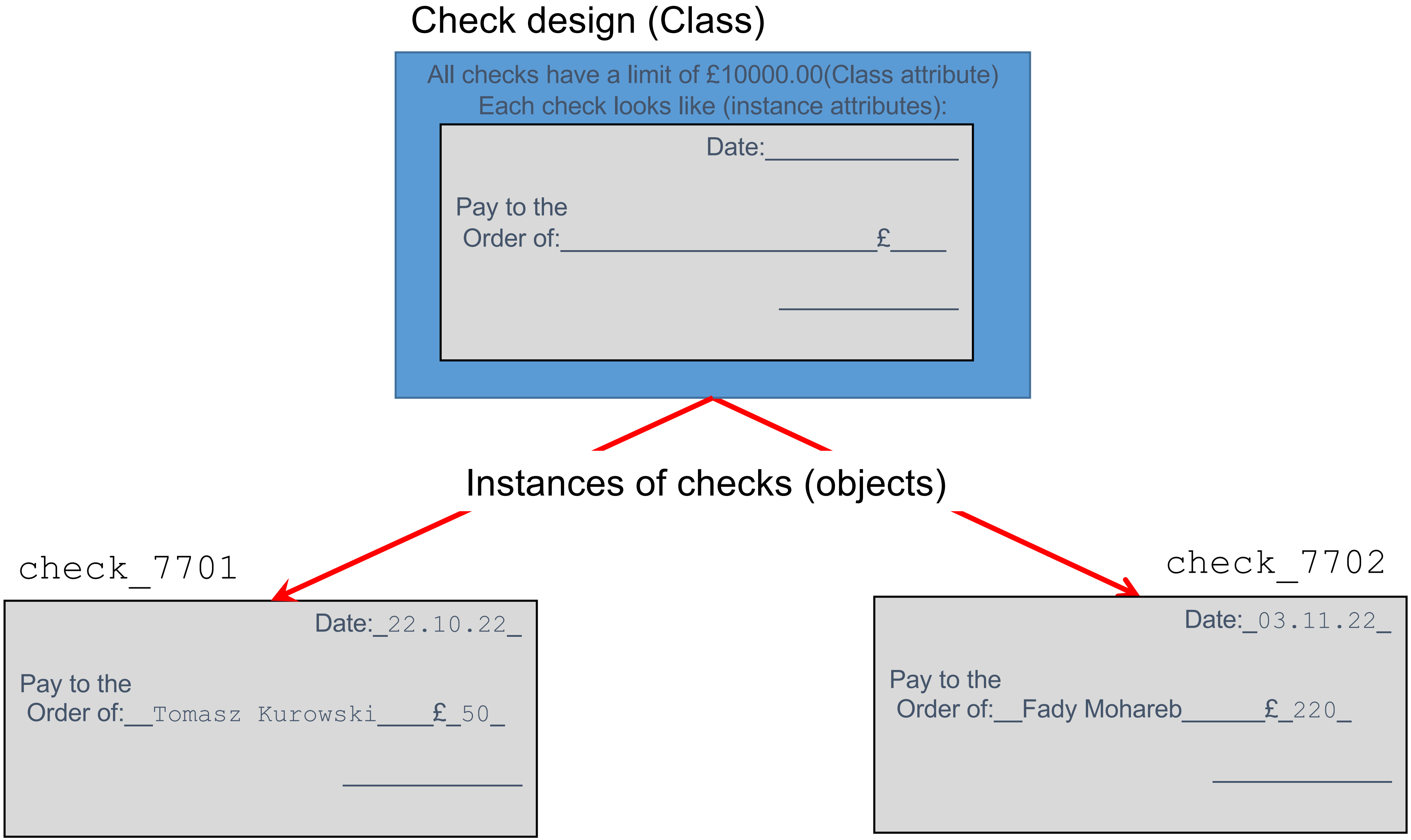
```
public class GermanShepherd{
    static int weightLimit = 40;
    static int heightLimit = 63;
    String name;
    String colour;
}
```

```
GermanShepherd myDog = new GermanShepherd(“Fred”, “brown”);
```



```
GermanShepherd neighbourDog = new GermanShepherd(“Dax”, “black”);
```

Class Attributes: Example_02



Check design (Class)

All checks have a limit of £10000.00(Class attribute)
Each check looks like (instance attributes):

Date:_______________

Pay to the
Order of:_________________________£____

____

Instances of checks (objects)

check_7701

Date:_22.10.22_

Pay to the
Order of:__Tomasz Kurowski____£_50_

____

check_7702

Date:_03.11.22_

Pay to the
Order of:__Fady Mohareb______£_220_

____

Class Attributes: Example_02 cont.

```
public class Check {
  static double limit = 10000.0; //Specific memory space is
    allocated
  String dateOfIssue; //no memory is allocated
  double amount;
  String payee;
 }
```

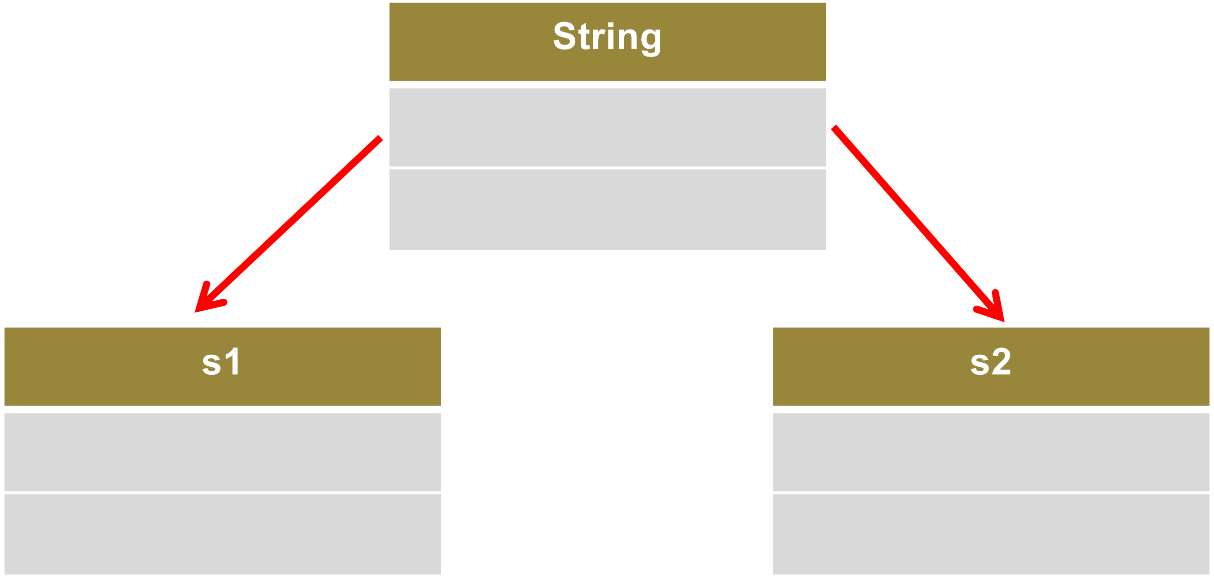
The `new` operator is responsible of allocating memory space for new instance declared:

```
Check check_7701 = new Check(“22.10.22”, “Tomasz Kurowski”,
  50.0);
```

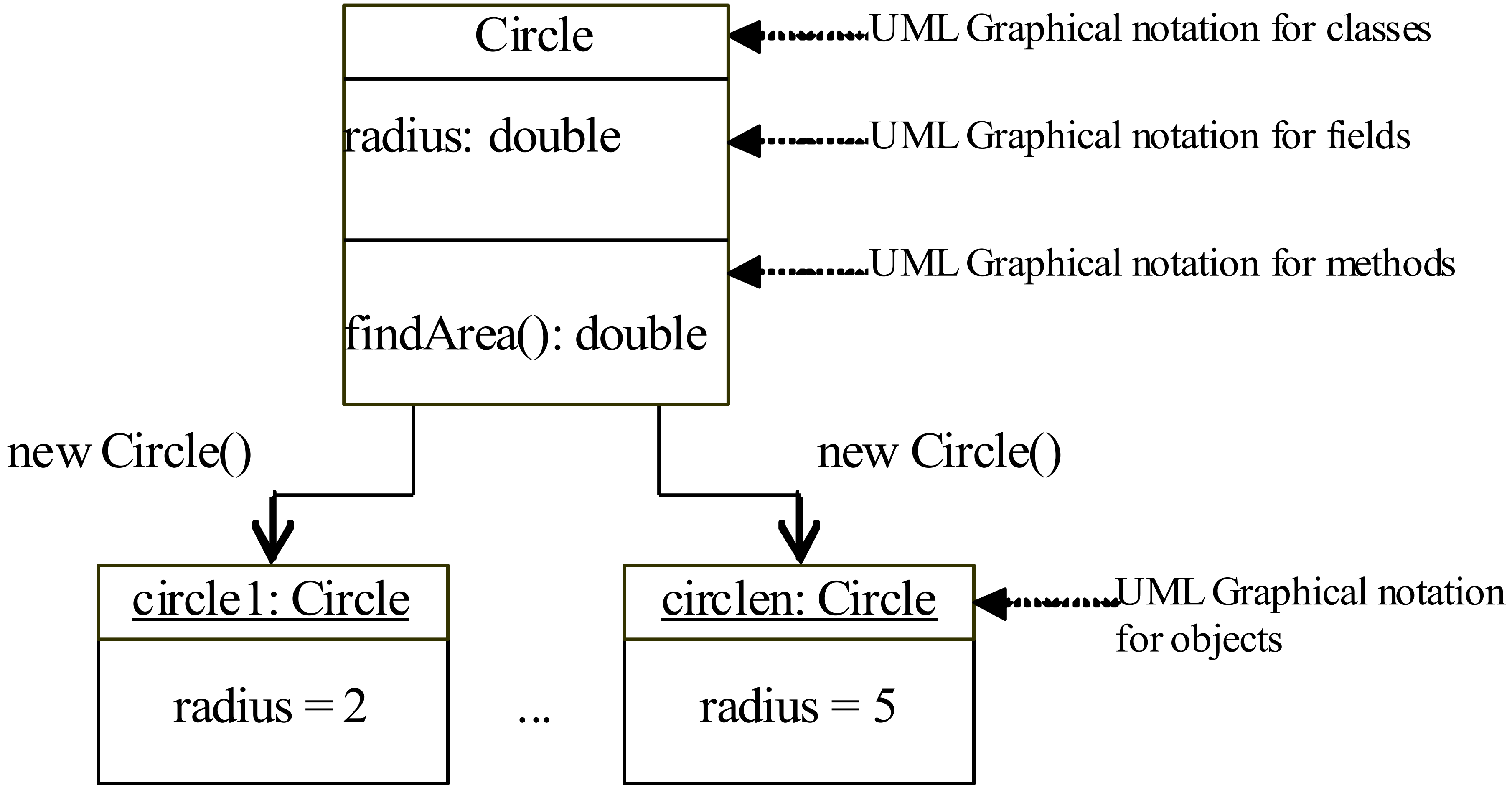
Unified Modelling Language (UML)

Unified Modelling Language

- Programmers like to use diagrams – we’ve seen Flowcharts
- Flowcharts show program flow – not representation of data
- Class diagrams are used to demonstrate relationships between classes and contents of classes
- Many ways of drawing these relationships – UML is a widely used standard. e.g to show Class relationships :



Class and Objects


Circle
radius: double
findArea(): double
UML Graphical notation for classes
UML Graphical notation for fields
UML Graphical notation for methods
new Circle()
new Circle()
circle1: Circle
radius = 2
...
circlen: Circle
radius = 5
UML Graphical notation for objects

(UML : Unified Modelling Language)

Class Declaration

```
class Circle {
  double radius = 1.0;        // Class wide scope

  double findArea(){
    return radius * radius * 3.14159;
  }
}
```

This is a ‘Template’ for some code and data. It does not exist until it is instantiated (created using ‘new’). Many separate copies can be created, all independent of each other.

Classes are another way for the software developer to break large complex problems down into self contained software components
Modern large applications are a collection of software components

Declaring Objects

Declaring Object Reference Variables :

Format : *ClassName objectName*;

Example: Circle myCircle;

Creating Objects

Format : *objectName = new ClassName();*

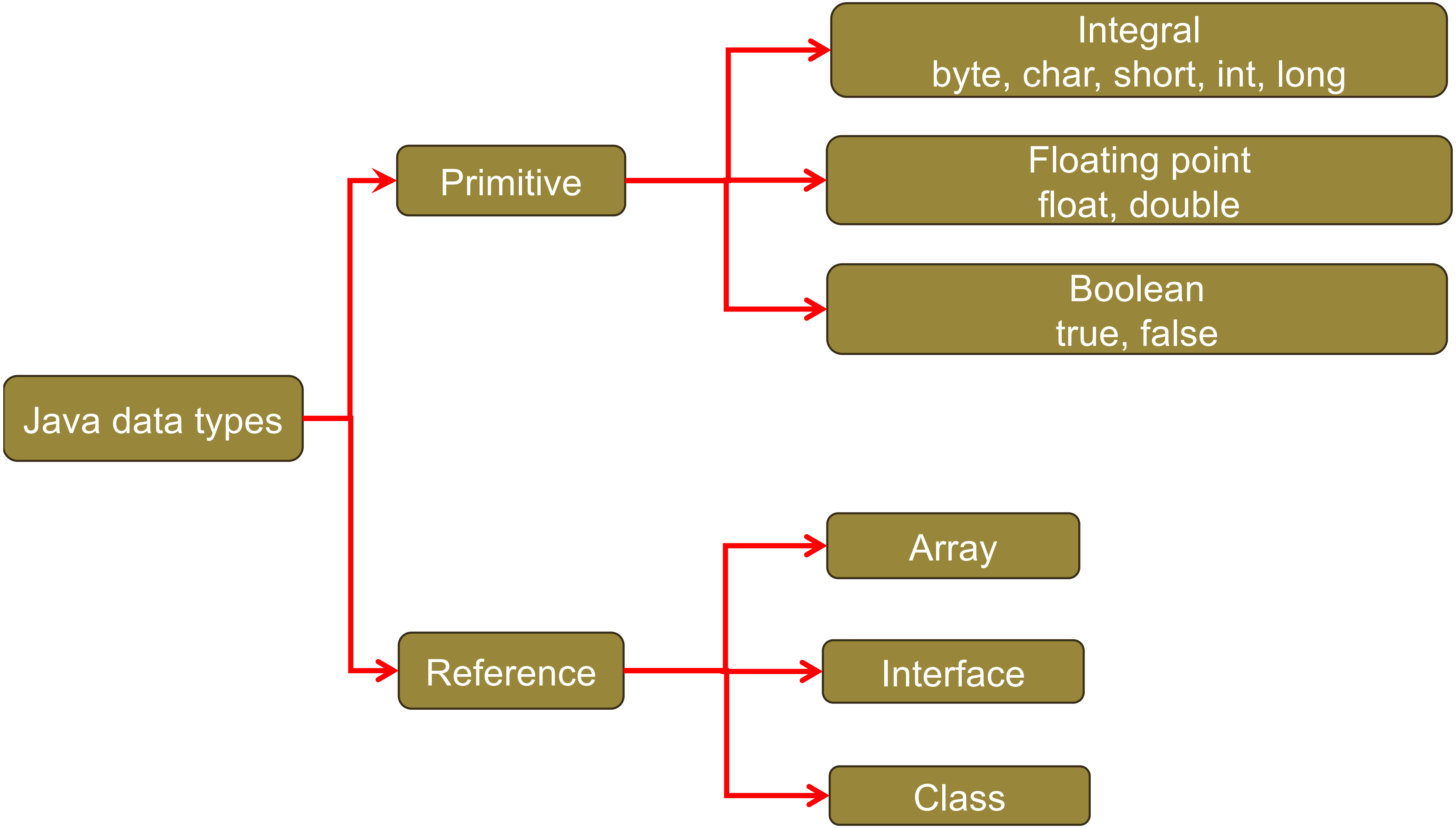
Example: myCircle = new Circle();

Declaring/Creating Objects in a Single Step

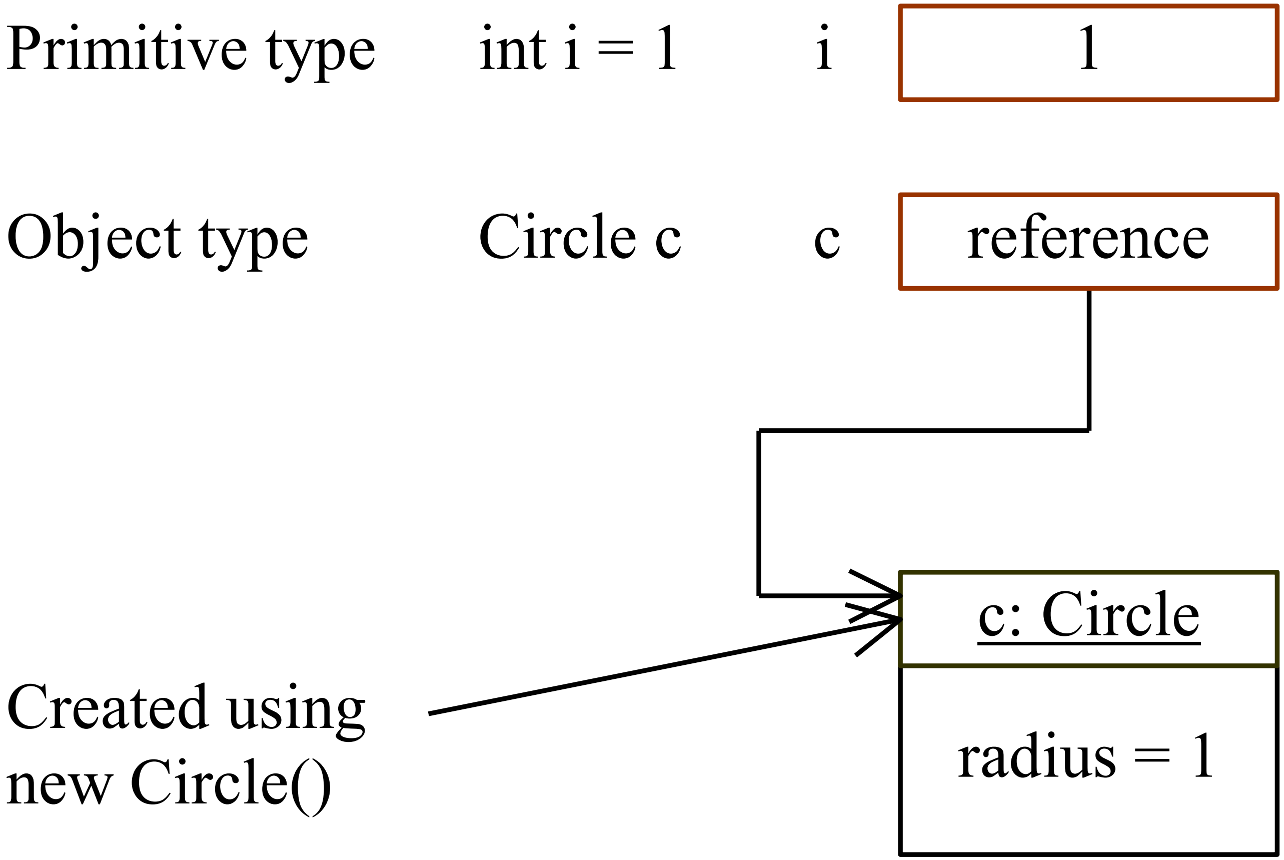
Format: *ClassName objectName = new ClassName();*

Example: Circle myCircle = new Circle();

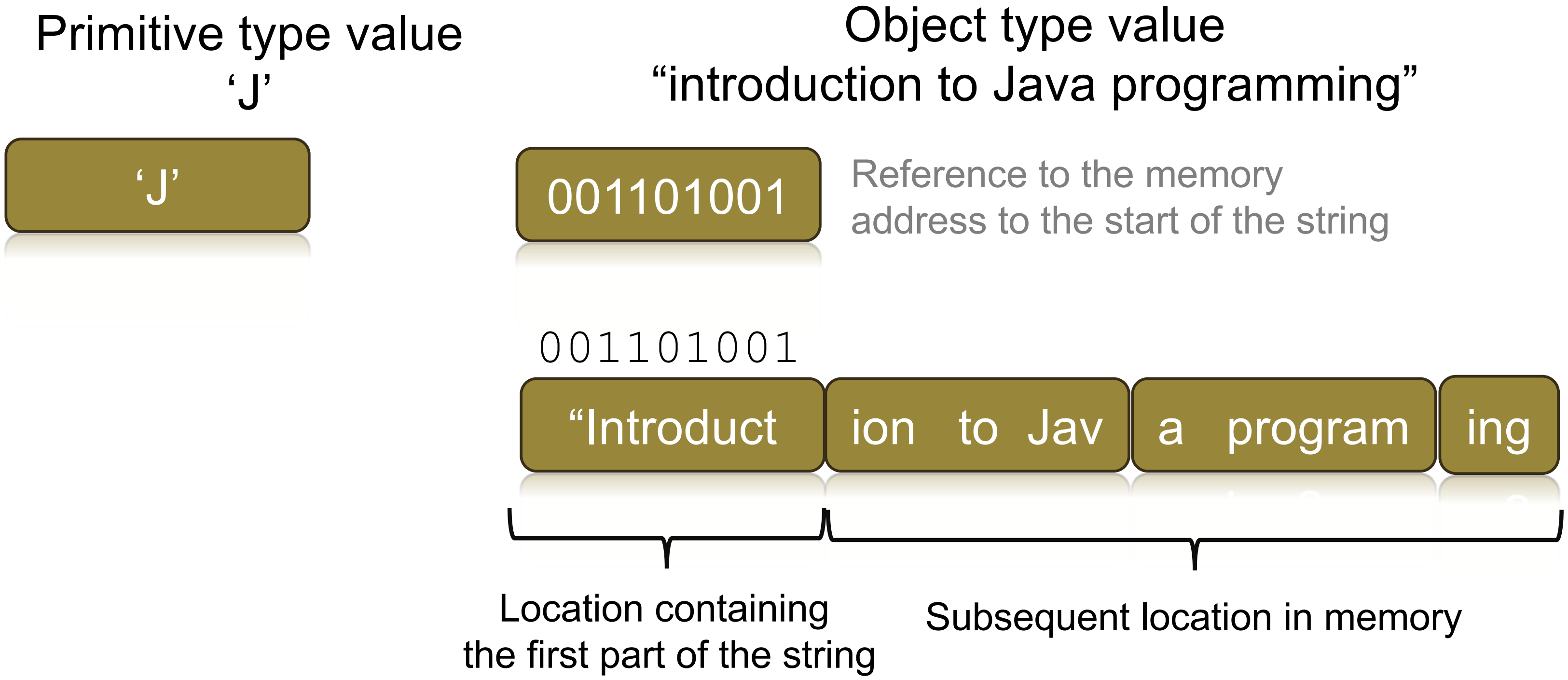
Java data types



Differences between variables of primitive Data types and object types



Differences between variables of primitive Data types and object types



Primitive type value
‘J’

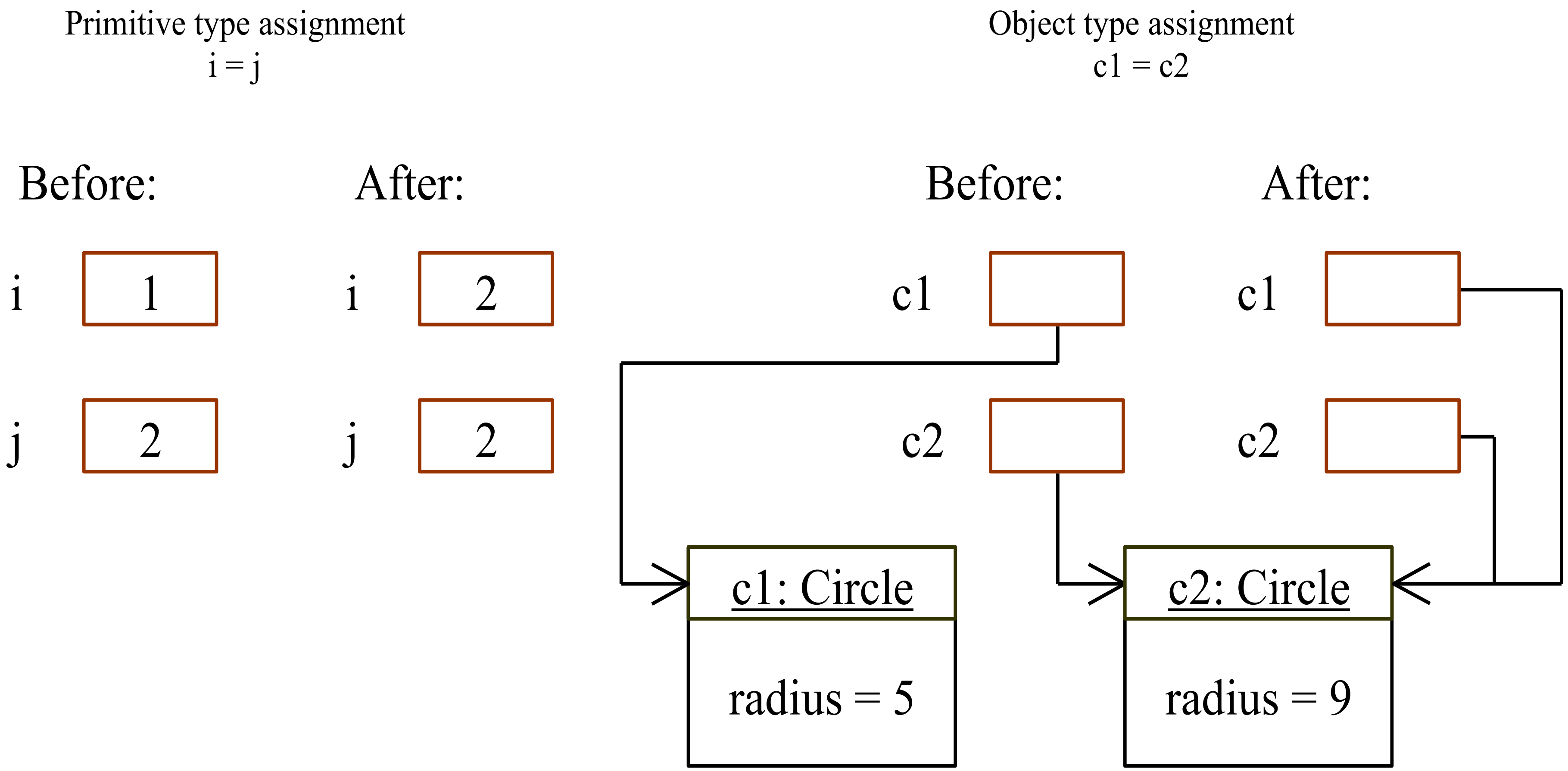
Object type value
“introduction to Java programming”

Reference to the memory address to the start of the string

Location containing the first part of the string

Subsequent location in memory

Copying Variables of Primitive
Data Types and Object Types



Garbage Collection

As shown in the previous figure, after the assignment statement c1 = c2, c1 points to the same object referenced by c2.

The object previously referenced by c1 is no longer useful. This object is known as garbage.

TAKIPI

Garbage is automatically collected by JVM.

Accessing Objects

- Referencing the object’s data:

```
objectName.data;
myCircle.radius;
```

- Invoking the object’s method:

```
objectName.method();
myCircle.findArea();
```

Example Using Objects

- Objective: Demonstrate creating objects, accessing data, and using methods.

TestCircle

Run

Example Using Objects

- Objective: Demonstrate creating objects, accessing data, and using methods.

TV

Run

Constructors

Constructors are a special kind of method that are invoked to construct objects – i.e. do the initialisation

```
Circle() {
  double radius = 1.0;
}
Circle C1 = new Circle()


Circle(double r) {
  radius = r;
}
myCircle = new Circle(5.0);
```

Constructors, cont.

A constructor with no parameters is referred to as a *default constructor*.

- Constructors must have the <u>same name</u> as the class itself.
- Constructors do <u>not</u> have a return type—not even void.
- Constructors are automatically called (invoked) when an object is created using the new operator
- Constructors perform the role of initializing objects.

Example Using Constructors

- Objective: Demonstrate the role of constructors and use them to create objects.

[TestCircleWithConstructors]

Run

Visibility Modifiers & Accessor Methods

By default, the class, variable, or data can be accessed by any class in the same package (public).

- **public**

  The class, data, or method is visible to any class in any package.

- **private**

  The data or methods can be accessed only by the declaring class.

Customary to provide ‘get’ and ‘set’ methods to read and modify private properties (variables).

Example using the `private` Modifier and Accessor Methods

In this example, private data are used for the radius and the accessor methods getRadius and setRadius are provided for the clients to retrieve and modify the radius.

Also known as ‘getter’ and ‘setter’ methods – some IDE’s define these automatically for you

CircleWithAccessors

TestCircleWithAccessors

Run (Netbeans)

Passing Objects to Methods

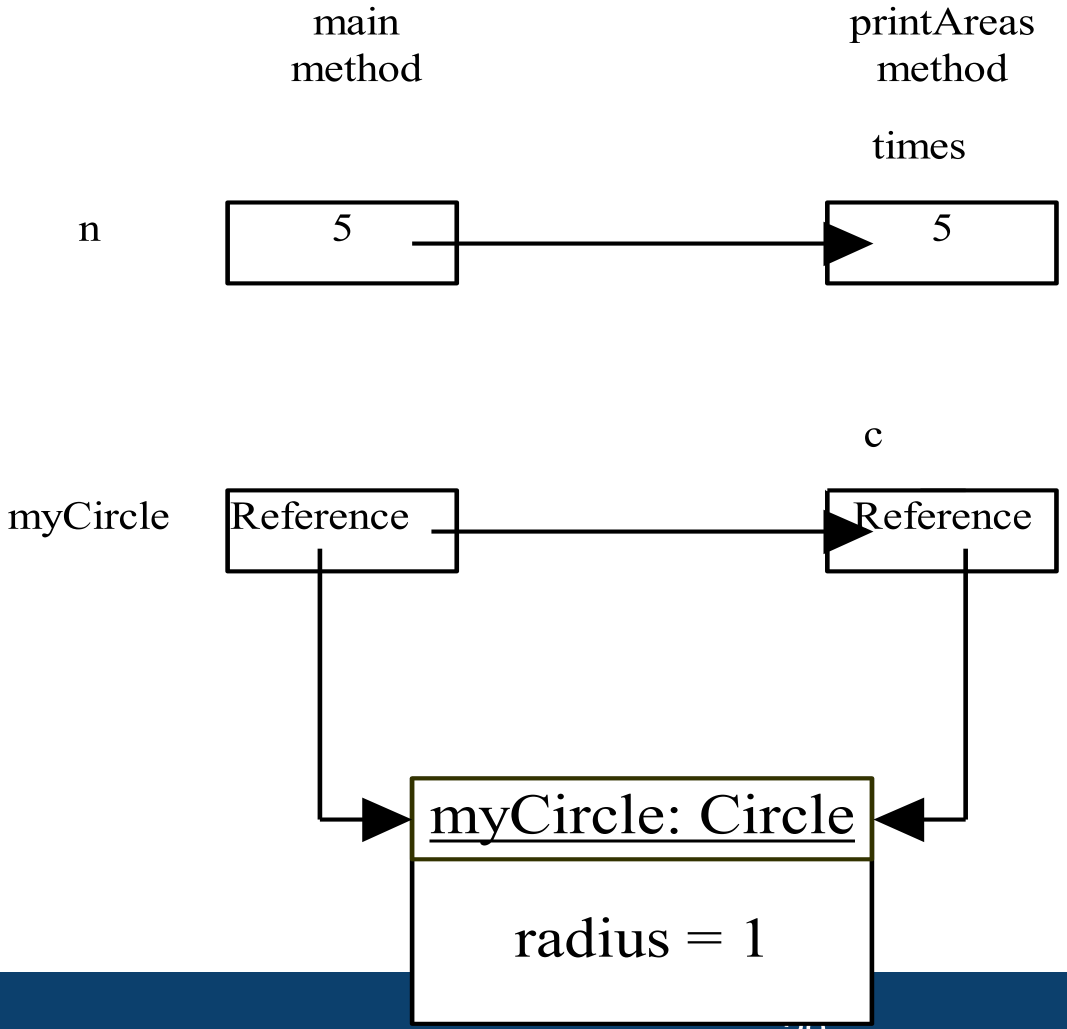
As always, passing by value – however, this time the value is a reference to the object

Example Passing Objects as Arguments

TestPassingObject (inCircleWithAccessors)

Run(Netbeans)

Passing Objects to Methods, cont.



Pass by value (here the value is 5)

Pass by value (here the value is the reference for the object)

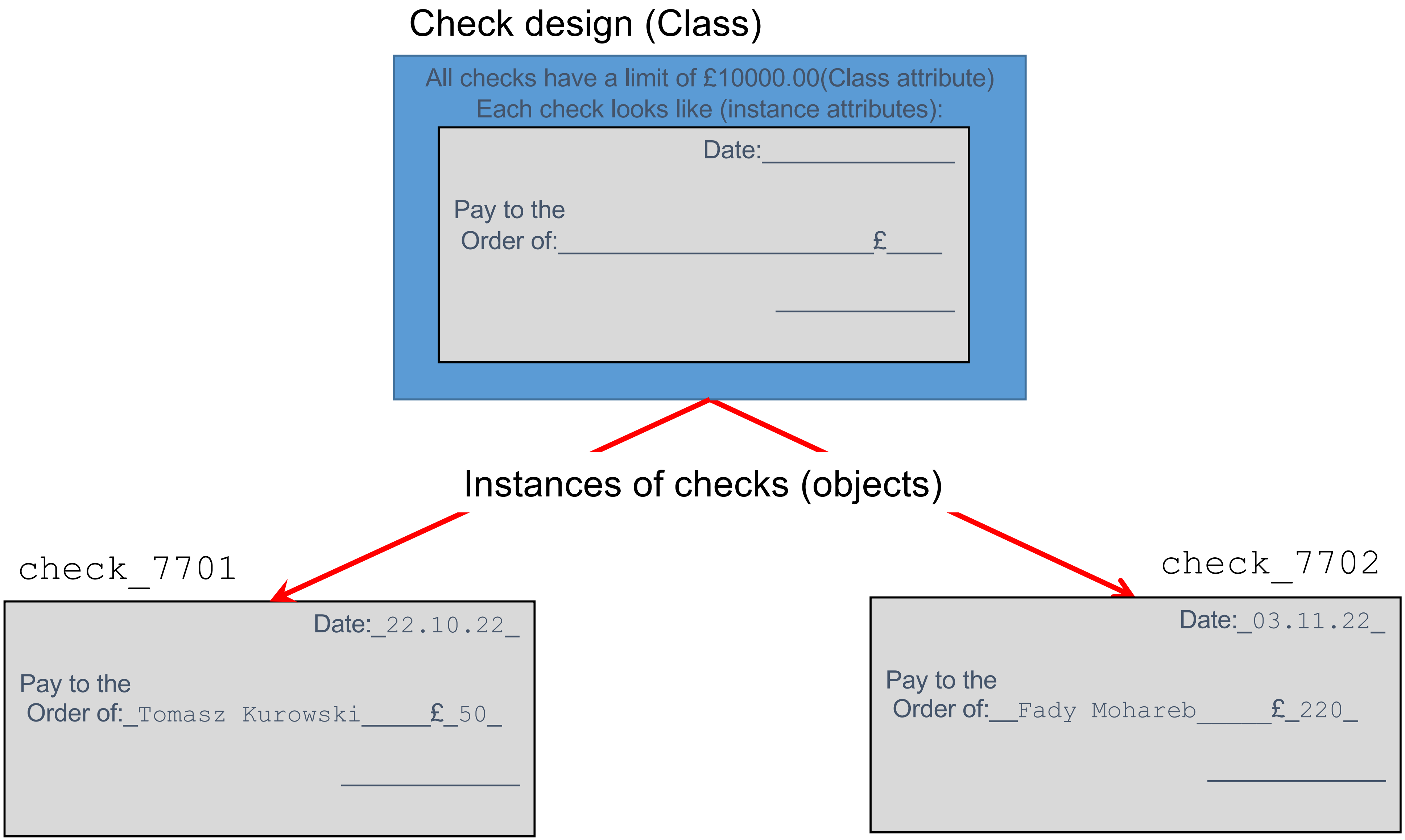
Instance Variables, and Methods

Instance variables belong to a specific instance of a class (Default).
Instance methods are invoked by an instance of the class.

Class Variables, Constants, and Methods

Class variables are shared by all the instances of the class.
Class methods are not tied to a specific object – they can be called without creating an instance of a class.
Class constants are final variables shared by all the instances of the class.
To declare class variables, constants, and methods, use the **static** modifier.

Quick reminder:



Check design (Class)

All checks have a limit of £10000.00(Class attribute)
Each check looks like (instance attributes):

Date:________________

Pay to the
Order of:__________________________£____

____

Instances of checks (objects)

check_7701

Date:_22.10.22_

Pay to the
Order of:_Tomasz Kurowski_____£_50_

____

check_7702

Date:_03.11.22_

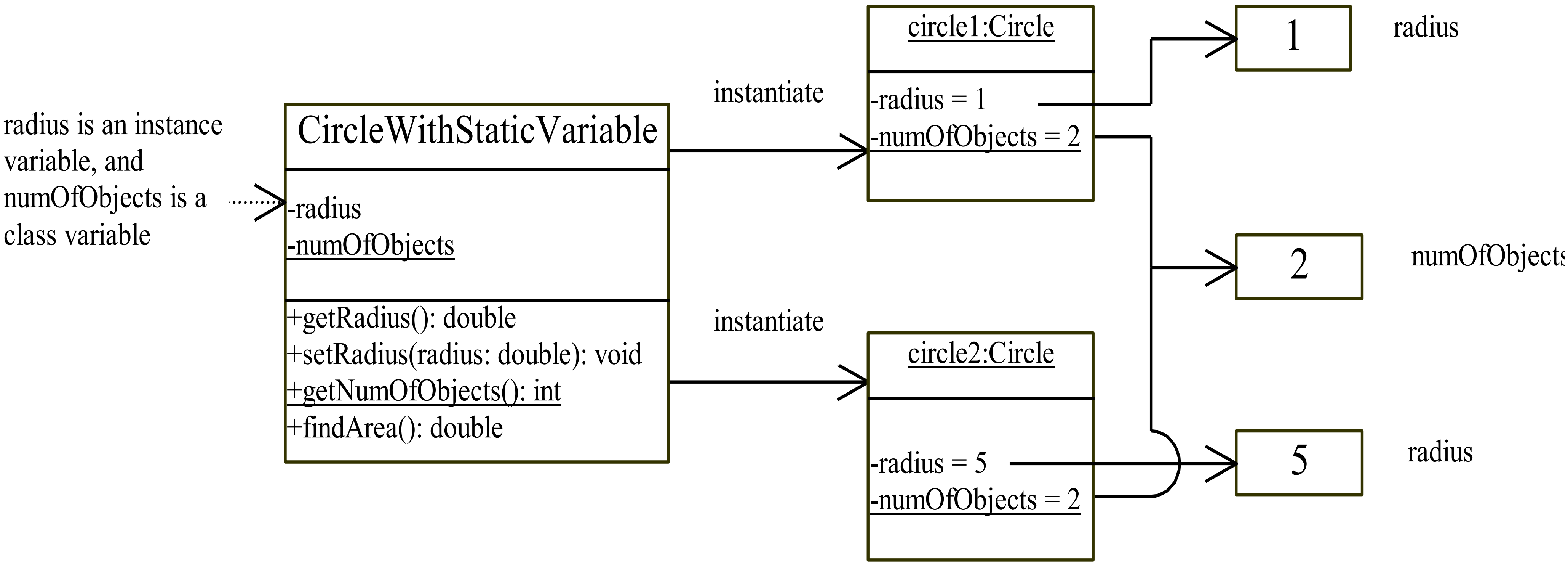
Pay to the
Order of:__Fady Mohareb_____£_220_

____

Class Variables, Constants, and Methods, cont.

UML Notation:
+: public variables or methods
-: private variables or methods
underline: static variables or metods

Memory



radius is an instance variable, and numOfObjects is a class variable

CircleWithStaticVariable

-radius
-numOfObjects

+getRadius(): double
+setRadius(radius: double): void
+getNumOfObjects(): int
+findArea(): double

instantiate

circle1:Circle

-radius = 1
-numOfObjects = 2

instantiate

circle2:Circle

-radius = 5
-numOfObjects = 2

Example Using Instance and Variables Class and Method

Objective: Demonstrate the roles of instance and class variables and their uses.

This example adds a class variable numOfObjects to track the number of Circle objects created.

Test CircleWithStaticVariable

Run (Netbeans)

Scope of Variables

- The scope of instance and class variables is the entire class. They can be declared anywhere inside a class.

```
Class Circle{
   double findArea(){
   return radius * radius * Math.PI;
 }
   Double radius = 1;
}
```

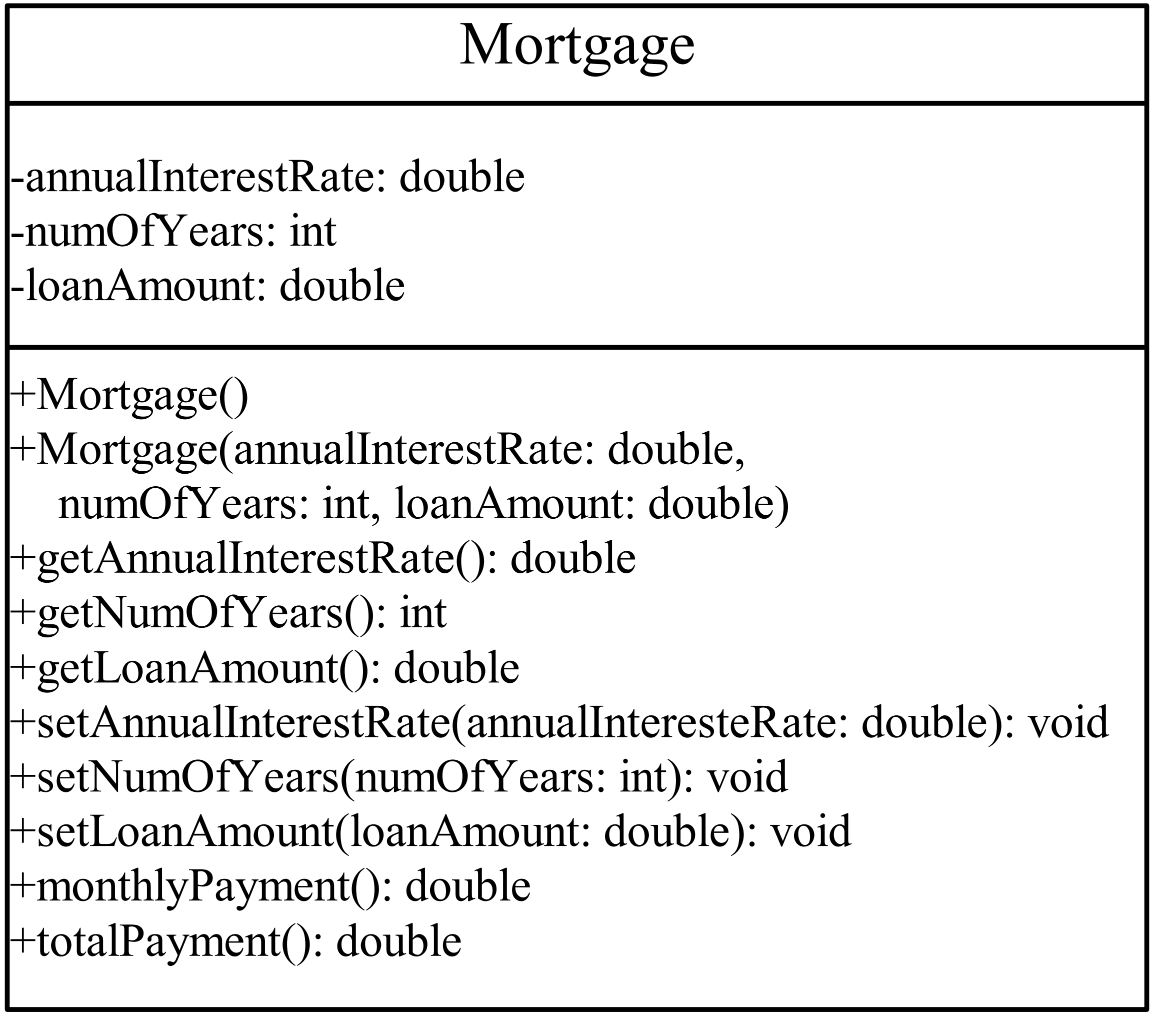
Scope of Variables

- The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable. A local variable must be declared before it can be used.

Class Abstraction

- Class abstraction means to separate the class implementation from the use of the class.
- The creator of the class provides a description of the class and let the user know how the class can be used.
- The user of the class does not need to know how the class is implemented.
- The detail of implementation is encapsulated and hidden from the user.

Example The Mortgage Class



Mortgage

-annualInterestRate: double
-numOfYears: int
-loanAmount: double

+Mortgage()
+Mortgage(annualInterestRate: double, numOfYears: int, loanAmount: double)
+getAnnualInterestRate(): double
+getNumOfYears(): int
+getLoanAmount(): double
+setAnnualInterestRate(annualInteresteRate: double): void
+setNumOfYears(numOfYears: int): void
+setLoanAmount(loanAmount: double): void
+monthlyPayment(): double
+totalPayment(): double

Mortgage

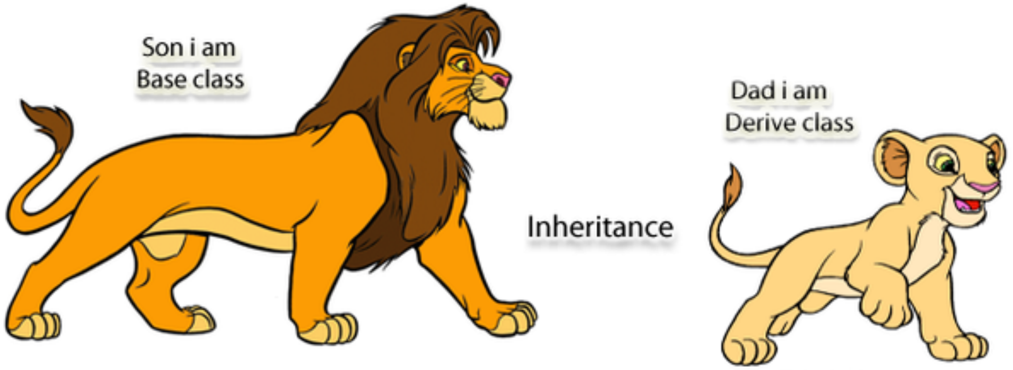
TestMortgageClass

Run

Class Inheritance and Interfaces

CLASS INHERITANCE AND INTERFACES

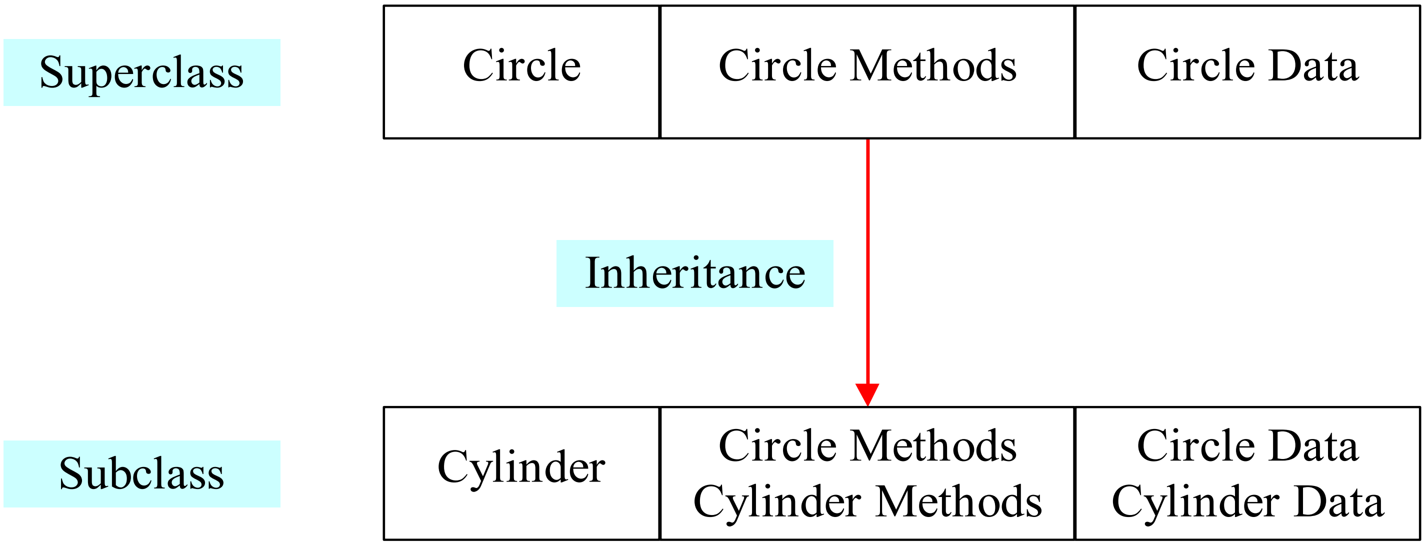
Inheritance

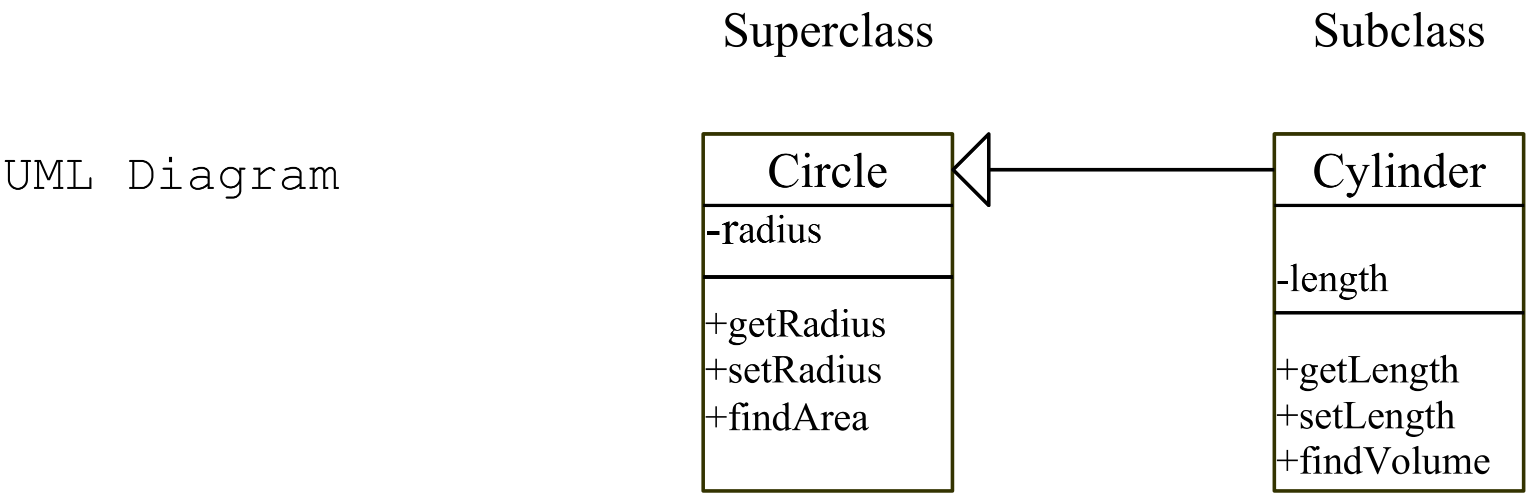


- New Classes can be created, based on other classes – Parents
- Parent class also known as ‘Base’ class, or Superclass.
- Child class also known as Derived class or Subclass.
- Child classes ‘inherit’ all the characteristics of parent class
- Child classes can use these Methods and Fields as they are, or can ‘override’ them with their own versions
- Child classes can add their own Methods and Fields.
- Swing library components are all inherited from ‘Component’ class
- Inheritance is useful when you want a number of different classes with many features in common. :

  Define one Baseclass with the common features, then derive Subclasses, adding Class specific detail to each.

Superclasses and Subclasses





Creating a Subclass

Creating a subclass extends the properties and methods from the superclass. You can also:

- Add new properties (length)
- Add new methods (findVolume)
- Override the methods of the superclass (findArea)

note - The `Cylinder` class overrides the `findArea()` method defined in the `Circle` class.

[Circle Class] [Cylinder Class]

The Keyword: this

- Sometimes you need to reference a class's hidden variable in a method.

```
Class Foo{
  int i = 5;

  public void setI(int i){
    this.i = i;
        }
   }
```

Using the Keyword `super`

The keyword super refers to the superclass of the class in which super appears.

It is used in a subclass to explicitly access items in the superclass. (Remember, a new object of the subclass type ‘inherits’ data and Methods from the superclass)

This keyword can be used in two ways:

- To call a superclass constructor
  - e.g. super (); super(parameters);
- To call a superclass method
  - e.g. super.method (parameters);

CAUTION

You must use the keyword super to call the superclass constructor, not the name of the superclass itself.

A constructor is used to construct an instance of a class. Unlike properties and methods, a superclass's constructors are not inherited in the subclass. They can only be invoked from the subclasses' constructors, using the keyword super.

If the keyword super is not explicitly used, the superclass's default constructor is automatically invoked.