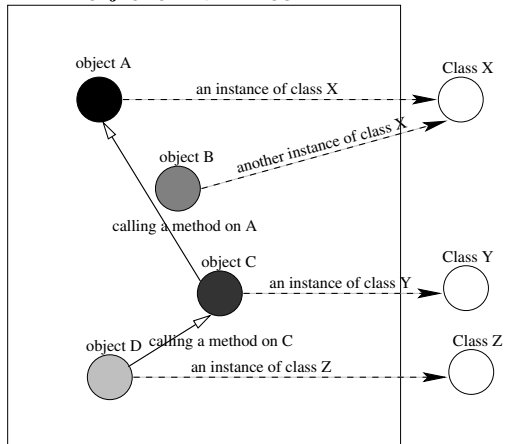


5COSC019W - OBJECT ORIENTED
PROGRAMMING
Lecture 2: More on Classes

Dr Dimitris C. Dracopoulos

OBJECT ORIENTED PROGRAM



Constructing Objects

Objects of a class are created by calling the *constructor* of the class with the `new` operator.

- ▶ Constructors are methods of the class and they always have the same name with the class itself.
- ▶ A class can have more than one constructors. Each one of them must have a different number (and/or type) of parameters.

Syntax:

```
new ClassName(parameters)
```

Constructor Example: The Rectangle Library class

The `Rectangle` class is defined in the library (package `java.awt`).

- ▶ A `Rectangle` object contains a set of numbers specifying an area in a coordinate space by its top-left point (x, y) , its width, and its height.

```
Rectangle r1 = new Rectangle(5, 10, 20, 30);
```

1. The constructor of the `Rectangle` class accepting four `ints` as parameters is called, to create a `Rectangle` object.
2. The created object has a top left corner at $(5, 10)$ and has a width equal to 20 and a height equal to 30.
3. The created object is returned and it is stored (its address) in variable `r1`.

An alternative constructor of the `Rectangle` class could be called to create an object:

```
Rectangle r2 = new Rectangle();
```

The no-arguments constructor, creates a `Rectangle` object whose top-left corner is at $(0, 0)$ in the coordinate space, and whose width and height are both zero

Packages

Classes (including library classes) belong to a *package*. Thus, a package is a collection of classes.

- ▶ To be able to use a class belonging to a package in a program, the class must first be imported in the program. There are two ways to do that:
 1. Import all the classes of the package into the program, e.g.

```
import java.awt.*;
```
 2. Import only the class that needs to be used, e.g.

```
import java.awt.Rectangle;
```

Example

```
import java.awt.Rectangle;

public class MoveTester {
    public static void main(String[] args) {
        Rectangle box = new Rectangle(5, 10, 20, 30);

        // Move the rectangle
        box.translate(15, 25);

        // Print information about the moved rectangle
        System.out.println("After moving, the top-left corner is:");
        System.out.println(box.getX() + ", " + box.getY());
    }
}
```

Implementing Classes

Implement a class which simulates a bank account. The following operations must be available (abstraction):

- ▶ Deposit money.
- ▶ Withdraw money.
- ▶ Get the current balance.


```
/**
    A bank account has a balance that can be changed by
    deposits and withdrawals.
*/
public class BankAccount {
    private double balance;

    /**
        Constructs a bank account with a zero balance.
    */
    public BankAccount()
    {
        balance = 0;
    }

    /**
        Constructs a bank account with a given balance.
        @param initialBalance the initial balance
    */
    public BankAccount(double initialBalance)
    {
        balance = initialBalance;
    }
}
```

```

    /**
     * Deposits money into the bank account.
     * @param amount the amount to deposit */
public void deposit(double amount)
{
    double newBalance = balance + amount;
    balance = newBalance;
}

    /**
     * Withdraws money from the bank account.
     * @param amount the amount to withdraw
     */
public void withdraw(double amount)
{
    double newBalance = balance - amount;
    balance = newBalance;
}

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

```

Testing a Class

```
/**
 * A class to test the BankAccount class.
 */
public class BankAccountTester {
    /**
     * Tests the methods of the BankAccount class.
     */
    public static void main(String[] args) {
        BankAccount harrysChecking = new BankAccount();
        harrysChecking.deposit(2000);
        harrysChecking.withdraw(500);
        System.out.println(harrysChecking.getBalance());

        BankAccount harrysSavings = new BankAccount(100);
        harrysSavings.withdraw(30);
        harrysSavings.withdraw(10);
        harrysSavings.deposit(20);
        double balance = harrysSavings.getBalance();
        System.out.println("Savings account balance: " + balance);
    }
}
```

When the above program is run, it displays:

1500.0

Savings account balance: 80.0

Overloading Methods

A class can have more than one methods with the same name, assuming that these methods have a different signature.

- ▶ The signature of a method consists of the combination of its name and its arguments (the specific order, number and type of arguments)

Example

```
public class Printer {
    int errorCode;

    // constructor 1
    public Printer() {
        System.out.println("Constructor with no arguments called!");
    }

    // constructor 2
    public Printer(int i) {
        errorCode = i;
        System.out.println("Constructor with an int argument called!");
    }

    public void display() {
        System.out.println(errorCode);
    }

    public void display(int i) {
        System.out.println(i);
    }

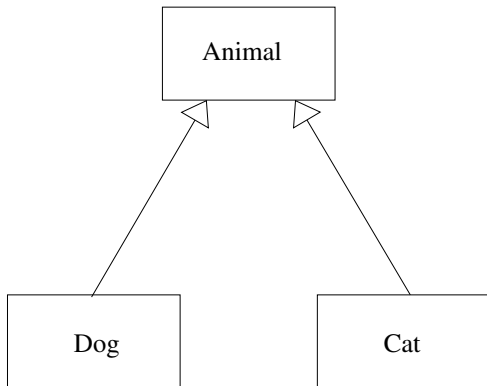
    public void display(String s) {
        System.out.println(s);
    }
}
```

```
public void display(int i, String s) {  
    System.out.println(i + s);  
}  
  
public void display(String s, int i) {  
    System.out.println(i + s);  
}  
}
```

Inheritance

Hierarchies of class can be created for the following reasons:

- ▶ Reusability of code (Reusability of the functionality of existing classes).
- ▶ Code is easier to maintain.
- ▶ Polymorphic behaviour (objects are manipulated via reference variables of the base class).



Example:

```
class Person {
    private String name;

    public Person() {

    }

    public Person(String name1) {
        name = name1;
    }

    // initialise the name instance field of the object
    public void setName(String name) {
        /* this is a shortcut for the object we are currently in.
           Thus, this.name is the instance field name within the
           current object */
        this.name = name;
    }

    // prints all information about the object
    public void info() {
        System.out.println("\nname: " + name);
    }
}
```



```

class Student extends Person {
    private String school;

    public Student(String school, String name) {
        this.school = school;
        setName(name);
    }

    // prints all information about the object
    public void info() {
        // call info() method of Person class
        super.info();
        System.out.println("school: " + school);
    }
}

```

```

class PostGraduateStudent extends Student {
    private String firstDegree; // what the first degree was on \pause

    public PostGraduateStudent(String school, String name,
                                String degree) {
        super(school, name); // call constructor of parent class
        firstDegree = degree; }

    public void info() {
        super.info();
        System.out.println("firstDegree: " + firstDegree);
    }
}

```

```
public class University {
    public static void main(String[] args) {
        Student s1 = new Student("IC", "John");
        Student s2 = new Student("MIT", "Helen");
        PostGraduateStudent s3 = new PostGraduateStudent(
                                                    "Westminster",
                                                    "George",
                                                    "music");

        s1.info();
        s2.info();
        s3.info();
    }
}
```

When the above program is run, it displays:

```
name: John
```

```
school: IC
```

```
name: Helen
```

```
school: MIT
```

```
name: George
```

```
school: Westminster
```

```
firstDegree: music
```

Overriding Methods

When a class defines a method with the same signature as a method in a parent class, the method is overridden in the subclass.

- ▶ `Student` overrides the `info` method with its own implementation (initially it inherits the `Person` version of `info`).
- ▶ `PostGraduateStudent` overrides the `info` method with its own implementation (initially it inherits the `Student` version of `info`).

The super keyword

The `super` keyword can be used for three purposes:

- ▶ To call a constructor of the parent class. In such a case, the `super` call must be the first statement in the constructor of the subclass:

```
public PostGraduateStudent(String school, String name, String degree) {  
    super(school, name);  
    firstDegree = degree;  
}
```

The first line in the above constructor calls the constructor `Student(String, String)` of the parent class.

- ▶ To call any method of the parent class, even if that method is overridden in the current class.
- ▶ To access a field of the parent class. For example, `super.x` in class B accesses the A instance field `x`:

```
class A {
    public int x;
}

class B extends A {
    public int x;

    void foo() {
        int y = super.x; // accesses x within A, NOT x in B!
    }
}
```