OBJECT ORIENTATED PROGRAMMING (OOP)
INHERITANCE Author Mr H Peuckert

PRISONER PROGRAM

Sometimes you want to add new fields to an object class but that leads to problems since you have to make chances to the class's code like its constructor method and toString method (where you have to add those new fields). You will also have to add more getter and setter methods to the class.

Let us look at the Prisoner program as an example:

1. Name of the prisoner.
2. Sentence they are serving in months.
3. Their prison block where their cell is to be found.

The Prisoner class is using the following class diagram:

Prisoner
Properties: - name : string - months : integer - block : character
Methods: + Constructor (n : string, m : integer, b : character) + getMonths : integer + getBlock() : character + toString() : string

Let us say some prisoners are on death row and we want to add the date and method of their execution, e.g. Bruce will be executed on 01/10/2019 and the method will be by *Lethal Injection.*

Let us say we want to store the values in the two fields (variables) called **execDate** and **execMethod**.
We can add the fields to the above class but then we then also have to add them to the constructor
method and to the toString method which will cause other problems.

We could go and **overload** the constructor method and create a second constructor method for the class:

+ Prisoner (String n, int m, char b, String ed, String em)

So a solution could be to add these two new fields to the class and some new methods:

Prisoner
Properties: - name : string - months : integer - block : character - **execDate : string** } *added these fields* **- execMethod : string**
Methods: + Constructor (n : string, m : integer, b : character) **+ Constructor (n : string, m : int, b : char, ed : string, em : string)** **+ getExecDate : string** } *added these* **+ getExecMethod : string** } *methods* + getMonths : integer + getBlock : character + toString() : string

Now we have **2 constructor methods** and the relevant one will be used when create an object of a prisoner that is not on death row (first constructor method) or a prisoner that is on death row (second constructor method).

But another problem will be created. Most prisoners are not on death row and the two fields (execDate and execMethod) will be created unnecessary for most objects (prisoners) wasting space in memory (RAM).

So the best solution will be **to just create a new class containing the new fields and methods**:

Prisoner
Properties: - name : string - months : integer - block : character
Methods: + Constructor (n : string, m : integer, b : character) + getMonths : integer + getBlock : character + toString : string

This arrow  **indicates a child class (sub class)**

DeathRow
Properties: - execDate : string - execMethod : string
Methods: + Constructor (n : string, m : integer, b : character, ed : string, em : string) + getExecDate : string + getExecMethod : string + toString : string

We have created a new class called DeathRow containing new fields and methods. We then say that the DeathRow class inherits from the Prisoner class - it inherits (uses) all the fields and methods of the *Prisoner* class. We can also say DeathRow is an extension of the Prisoner class. DeathRow also uses the fields and methods of the Prisoner class. The new toString method of DeathRow **overrides** the toString method of Prisoner (when working with the DeathRow class it will use its own version of toString, not the inherited version)

We say Prisoner is the super class and Deathrow is the subclass.
OR: We say Prisoner is the parent class and Deathrow is the child class.

To create a new instance of the Prisoner class will be as always (create/instantiate an object of the Prisoner class):

Prisoner inmate1 = new Prisoner("Ray",5,'A');

To create an object (instance) of the DeathRow class will look similar but you will sent more data to the constructor method (go and look at the constructor method in the above class diagram to see what data is needed and in what order it should be in). Ray is not on death row so we create an object of the Prisoner class (done above). Bruce is on death row so we create an object of the DeathRow class.

DeathRow inmate2 = new DeathRow("Bruce",359,'C',"31/12/2017","Lethal Injection");

This is not a perfect solution because if we change the super class we have to make changes to sub class. The sub class is very dependent on the super class.

Advantages of Inheritance:

- One of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses. Where equivalent code exists in two related classes, the hierarchy can usually be refactored to move the common code up to a mutual superclass. This also tends to result in a better organization of code and smaller, simpler compilation units.
  Inheritance can also make application code more flexible to change because classes that inherit from a common superclass can be used interchangeably.
- Reusability – You can use public methods of the super class without rewriting the same code in the sub class.
- Data hiding - super class can decide to keep some data private so that it cannot be altered by the sub class.
- Overriding - With inheritance, we will be able to override the methods of the super class so that meaningful implementation of the sub class method can be designed in the subclass.

More definitions:

- Inheritance refers to a feature of Java programming that lets you create classes that are derived from other classes. A class that's based on another class inherits the other class. The class that is inherited is the parent class, the base class, or the superclass. The class that does the inheriting is the child class, the derived class, or the subclass.
- A subclass automatically takes on all the behavior and attributes of its base class. Thus, if you need to create several classes to describe types that aren't identical but have many features in common, you can create a base class that defines all the common features. Then you can create subclasses that inherit the common features.
- A subclass can add features to the base class it inherits by defining its own methods and fields. This is one of the ways a derived class distinguishes itself from its base class.
- The ability of a subclass to override a method allows a class to inherit from a superclass whose behaviour is "close enough" and then to modify behaviour as needed. The overriding method has the same name, number and type of parameters, and return type as the method that it overrides.
- Rules for method overriding: The argument list should be exactly the same as that of the overridden method. The return type should be the same or a subtype of the return type declared in the original overridden method in the superclass. Think of the toString method in the sub class and super class. When running customer2.toString() it will call the toString method of the sub class (and not the super class) – it overrides the other toString method.

Practical Example:

In the following program we use the example of the Prisoner class and the DeathRow class who inherits from the Prisoner class (as discussed above):

```
package inheritance_prisoner;

public class Prisoner {                          ← Super class / Parent class

   private String name;
   private int months;
   private char block;

   public Prisoner(String n, int m, char b) {
      name = n;
      months = m;
      block = b;
   }

   public String getName() {
      return name;
   }

   public int getMonths() {
      return months;
   }

   public char getBlock() {
      return block;
   }

   public String toString() {
      return block + "\t" + name + "\t(" + months + ")";
   }

}
```

```
package inheritance_prisoner;

public class DeathRow extends Prisoner {         ← Sub class / Child class

   private String execDate;
   private String execMethod;

   public DeathRow(String n, int m, char b, String ed, String em) {
      super(n, m, b);          // This will call (go to) the constructor
      execDate = ed;           // method of the super class. So you can
read
```

```
    execMethod = em;                                    //  it as: Prisoner(n,m,b);
  }

  public String getExecDate() {
    return execDate;
  }

  public String getExecMethod() {
    return execMethod;
  }

  public String toString() {
    return super.toString() + "\t" + execDate + "\t" + execMethod;
  }
                  // This means call (go to) the toString method of the Prisoner class (super class)
}
```

```
import javax.swing.JOptionPane;

public class PrisonerUI {                    ← User Interface class (that contains a main method)

   public static void main(String[] args) {

      Prisoner prisoner1;

      prisoner1 = new Prisoner("Trix", 11, 'A');

      DeathRow prisoner2;

      prisoner2 = new DeathRow("Zack", 60, 'C', "29/05/2018", "Hanging");
      System.out.println(prisoner1.getName() + " is serving " + prisoner1.getMonths() + " months.");

      System.out.println(prisoner2.getName() + " is serving " + prisoner2.getMonths() + " months.");

      System.out.println("\nPrisoner 1: " + prisoner1.toString());
      System.out.println("\nPrisoner 2: " + prisoner2.toString());
      System.out.println("");                  // This read as: "Is prisoner1 an object
                                                // of the DeathRow class?"
      if (prisoner1 instanceof DeathRow) {
         System.out.println(prisoner1.getName() + " is instance of the DeathRow class.");
      } else {
         System.out.println(prisoner1.getName() + " is instance of the Prisoner class.");
      }

      System.out.println("");

      if (prisoner2 instanceof DeathRow) {
         System.out.println(prisoner2.getName() + " is instance of the DeathRow class.");
      } else {
         System.out.println(prisoner2.getName() + " is instance of the Prisoner class.");
      }
   }

}
```

The **output** of the above program:

```
Trix is serving 11 months.
Zack is serving 60 months.
Prisoner 1: A  Trix      (11)
Prisoner 2: C  Zack      (60)    29/05/2018    Hanging
Trix is instance of the Prisoner class.
Zack is instance of the DeathRow class.
```