Object Oriented Programming – SCJ2153

Inheritance

Associate Prof. Dr. Norazah Yusof

Introduction to Inheritance

- *Inheritance* is one of the main techniques of object-oriented programming (OOP)
- Using this technique, a very general form of a class is first defined and compiled, and then more specialized versions of the class are defined by adding instance variables and methods
  - The specialized classes are said to *inherit* the methods and instance variables of the general class

The "is a" Relationship

- The relationship between a superclass and an inherited class is called an "is a" relationship.
  - A student "is a" person.
  - A cow "is a" animal.
  - A bicycle "is a" vehicle.
- A specialized object has:
  - all of the characteristics of the general object, plus
  - additional characteristics that make it special.
- In object-oriented programming, *inheritance* is used to create an "is a" relationship among classes.

The “is a” Relationship

- We can *extend* the capabilities of a class.
- Inheritance involves a superclass and a subclass.
  - The *superclass* is the general class and
  - the *subclass* is the specialized class.
- The subclass is based on, or extended from, the superclass.
  - Superclasses are also called *base classes*, and
  - subclasses are also called *derived classes.*
- The relationship of classes can be thought of as *parent classes* and *child classes*.

Inherited Members

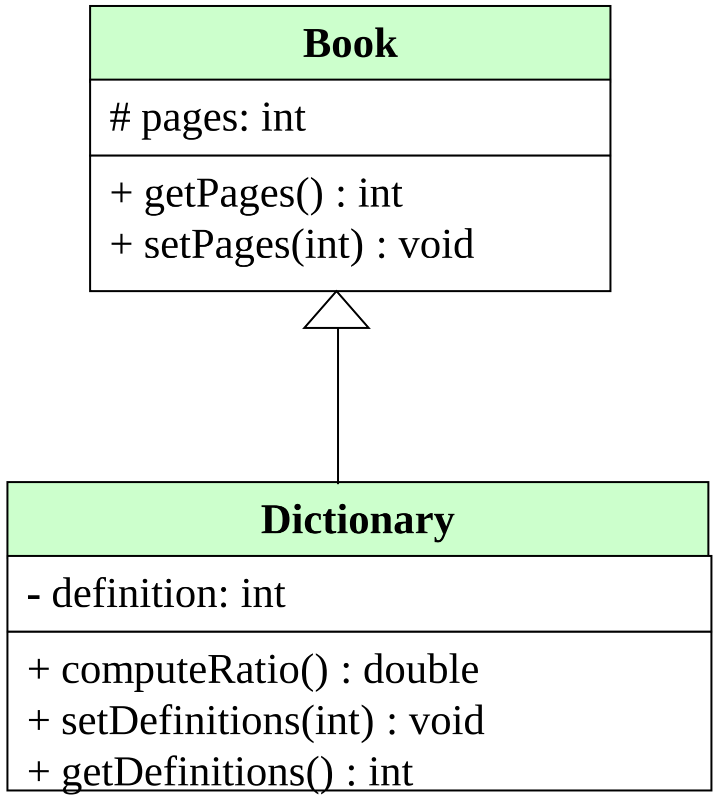
- A derived class automatically has all the instance variables, all the static variables, and all the public methods of the base class
  - Members from the base class are said to be *inherited*
- Definitions for the inherited variables and methods do not appear in the derived class
  - The code is reused without having to explicitly copy it, unless the creator of the derived class redefines one or more of the base class methods

Protected Modifier

- **Protected modifier is used to control access to the members of a class.**
- A variable or method declared with protected visibility may be accessed by any class in the same package. In other words a derived class can reference it.
- A protected visibility allows a class to retain some encapsulation properties. However, the encapsulation is not as tight as private.

Example Derived Classes

- An arrow with an open arrowhead is used to show inheritance.
- The symbol # is used to represent protected access.



Example Derived Classes

- When a derived class is defined, it is said to inherit the instance variables and methods of the base class that it extends
  - Class **`Book`** defines the instance variables **`pages`** in its class definition
  - Class **`Dictionary`** also has these instance variables, but they are not specified in its class definition.
  - Class **`Dictionary`** has additional instance variable **`definitions`** that are specified in its class definition.

Example Derived Classes

- Just as it inherits the instance variables of the class **`Book`**, the class **`Dictionary`** inherits all of its methods as well
  - The class **`Dictionary`** inherits the methods **`getPages`**, and **`setPages`** from the class **`Book`**
  - Any object of the class **`Dictionary`** can invoke one of these methods, just like any other method.

Programming Example

Program description

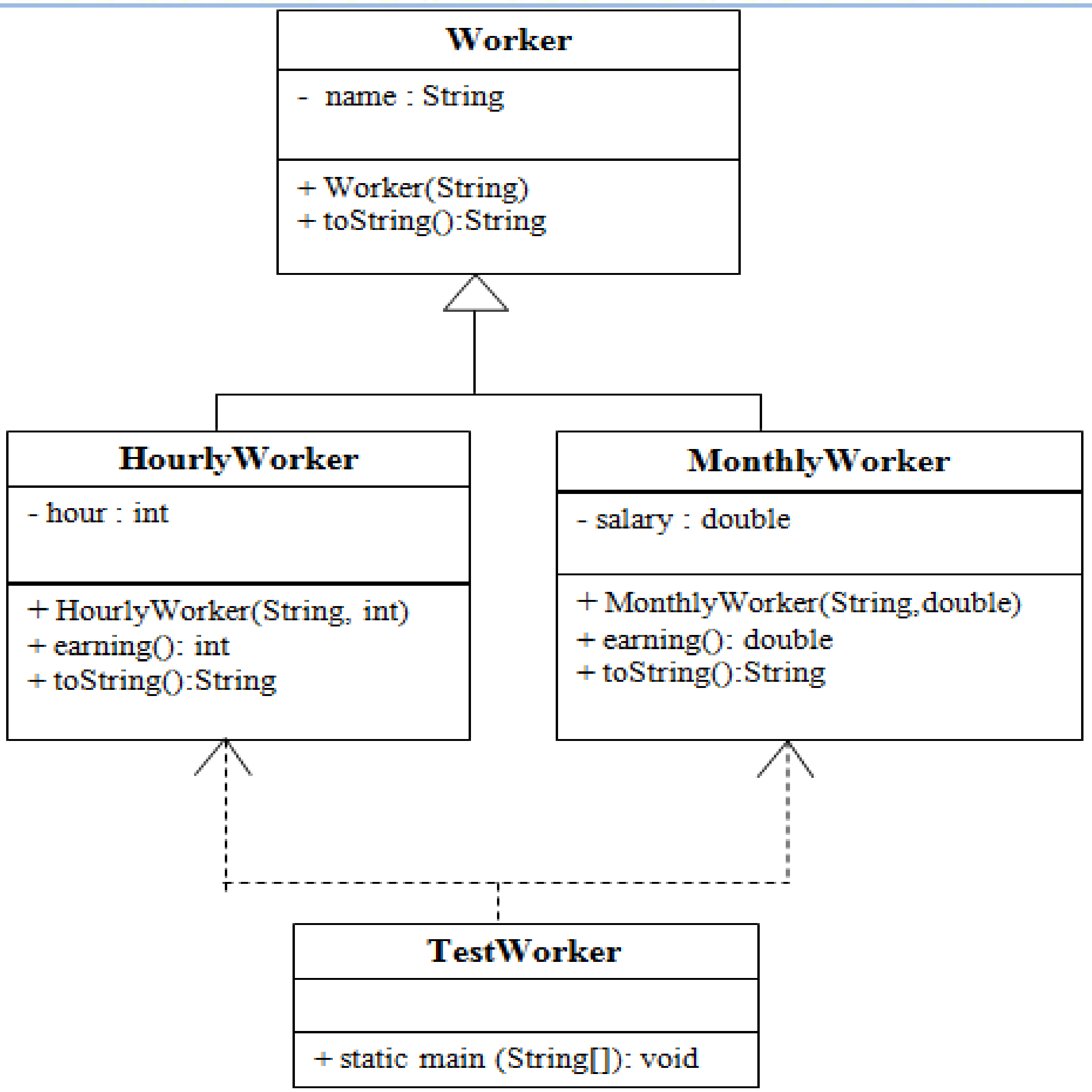
- Write a complete Java program that contains classes as shown below:
  - The **Worker** class serves as the parent of all classes and contains information that applies to all workers. Each worker has a name of type String. The **Worker** class contains a **toString** method to return the name of the worker.

Program description

– The **HourlyWorker** class represents a worker that earns an income based on hourly worked. The HourlyWorker class contains of an instance variable i.e. hour of type integer that represents the number of hours worked. It consists of an **earning** method that returns the result of hour * 5.00 (rate per hour). It also consists of a **toString** method to return the worker information and his/her income.

Program description

- The **MonthlyWorker** class represents a worker that earns an income in a monthly basis. It contains an instance variable i.e. salary of type double that represents the monthly salary. It consists of an **earning** method that returns the salary. It also contains a **toString** method to return the worker information and his/her monthly income.
- The **TestWorker** class represents the application program that instantiates **HourlyWorker** and **MonthlyWorker** objects, and then invoke their **toString()** method.


Worker
- name : String
+ Worker(String)
+ toString():String
HourlyWorker
- hour : int
+ HourlyWorker(String, int)
+ earning(): int
+ toString():String
MonthlyWorker
- salary : double
+ MonthlyWorker(String,double)
+ earning(): double
+ toString():String
TestWorker
+ static main (String[]): void

TestWorker.java

```
class Worker {
  private String name;

  public Worker(String n) {
       name = n;
  }

  public String toString(){
        return "Name :"+name;
  }
}

class HourlyWorker extends Worker {
      private int hour;

      public HourlyWorker(String n, int h) {
            super(n);
            hour = h;
      }
```

TestWorker.java

```
        public double earning() {
                return hour * 5.00;
        }
     public String toString(){
        return super.toString()+"\nHours worked:" + hour+
            "\nEarning:"+earning();
   }
}

class MonthlyWorker extends Worker {
     private double salary;

        public MonthlyWorker(String n, double s) {
                super(n);
                salary = s;
        }

        public double earning() {
                return salary;
        }
```

TestWorker.java

```java
public String toString(){
    return super.toString() + "\nEarning:"+ earning();
  }
}

public class TestWorker {
  public static void main(String[] args){
    HourlyWorker h1 = new HourlyWorker("Ali",6);
    System.out.println (h1.toString());

    MonthlyWorker h2 = new MonthlyWorker("Abu",6000);
    System.out.println (h2.toString());
  }
}
```