Introduction to Programming

14. Types, Classes and Objects

Part 1

Program Execution in Java

- For Java applications, execution of a program involves executing the statements in the main() method
  - For this module, we will ignore threads
- The main() method may call other methods to execute the statements that they contain
  - These methods in turn may call other methods
  - When execution of the statements in a method are finished control returns to the point in the calling method immediately following the call

Applications are Classes

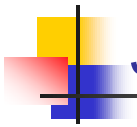
- In Java it is not possible to have a main() method, or any method, that is not part of a class
- One of the roles of classes in Java is to serve as applications by providing a main() method
- The main() method is always a static method

The Roles of Java Classes

- In addition to serving as applications, classes in Java have two other roles
  - A Java class can be a container for static methods and variables
    - TextIO and Math are examples of such classes
  - A Java class allows you to define a *type*
    - String is an example of such a class

Types revisited

- A *type* is a set of values, and the set of things that you can do with those values (the operations that can be applied to them)
- For the primitive types and for the type, String, the values can be written directly in your code using literals (*e.g.* 1, 3.14, **false**, "Hello") but for most types this is not the case
  - The vast majority of types in Java are declared using classes



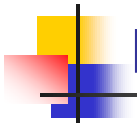
Java classes

- A class declaration defines a new *reference type*
- The class name can be used to declare variables of that reference type (just as the primitive type names can be used to declare variables of primitive types)
- The values stored in reference type variables are references that allow access to instances of the class

Example

- Simplified Student record (see textbook page 192) – holds data about students on a course with three tests

```
public class Student {
    public String name; // Student's name.
    public double test1, test2, test3; /* Grades on
                                          three tests. */
   // Calculate the average test grade
    public double getAverage() {
        return (test1 + test2 + test3) / 3;
    }
} // end of class Student
```

Notes on Student Example

- No variables or methods in the Student class are declared as **static**
  - So the only purpose of the class is to create Student objects
- Each Student object (or instance) has its own name and three test grades
  - The variables that hold this information are called ***instance variables***
- Each Student has its own method to calculate its average grade
  - Note the method has no parameters but directly accesses the three instance test grade variables
  - getAverage() is an example of an ***instance method***

Students as Objects

- In object-oriented programming, an *object* has
  - State – this is described by the values of its instance variables
  - Behaviour – this is described by the methods that it defines
    - Calling the methods may or may not change the state of the object
- One can think of each instance of the class Student as an object
  - currently a poorly designed one (other classes can change a Student's state without calling its methods, so the Student class has no control over the state of its instances)

Constructors

- Are used to create instances of classes, and to initialise the state of the instance
- Like a method, but has no return type and must have the same name as the class
- A class can have several constructors as long as the argument list is different for each (overloading)
- The compiler will create a constructor with no arguments for any class that does not declare any constructors
  - Constructor with no arguments is called a "default constructor"
  - The compiler provides the Student class with a default constructor as the class does not declare one

Constructors continued...

- If you could see the constructor for the Student class it would look something like this:

```
public Student() {  // the default constructor
   super(); // will come back to this in a
            // later lecture
   name = null;
   test1 = 0.0;
   test2 = 0.0;
   test3 = 0.0;
}
```


Initializing the instance variables

- As with the elements of an array, instance variables are automatically initialised when the instance is created
  - test1, test2, test3 are all initialised to 0.0
    - Any other primitive numeric fields would also be set to zero
    - Any **boolean** fields would be set to **false**
  - name, a variable belonging to a reference type, is initialised to **null**

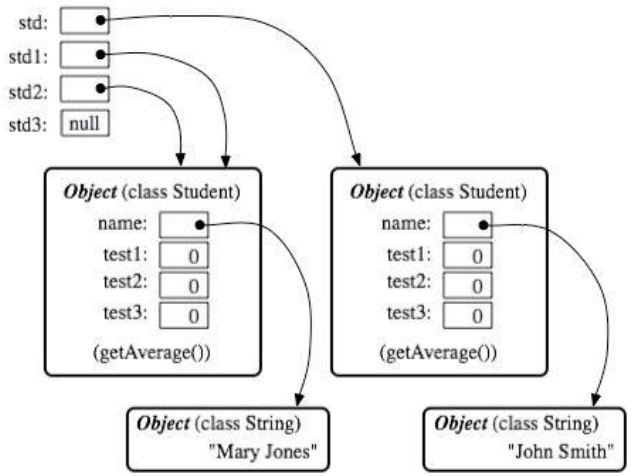
Using the Student class

- Here is a Java application that uses the Student class (see textbook page 194)

```
public class Application {
  public static void main(String[] args) {
    Student std, std1, std2, std3;// declare Student variables
    std = new Student(); /* create a Student object and store
                            the reference to it in std */
    std1 = new Student(); // create a second Student object
    std2 = std1; // std2 and std1 both refer to second Student
    std3 = null; // store a null reference in std3
    std.name = "John Smith";
    std1.name = "Mary Jones";
  }
}
```


Effect of running Application

(after page 195 of textbook, each object is an instance of its class)



Notes on Example

- Application calls the Student constructor to create two Student objects
- The value of std3 is **null**
  - You can check if a variable equals **null**

```
if (std3 == null) { ... }
```

  - It is a runtime error if you attempt to access an object using a **null** reference

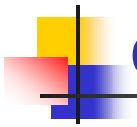
```
std3.name = "John Jones";
        // NullPointerException!
```


Notes on reference types...

- For reference types (objects) in general
  - The value of any variable that is not of a primitive type is either **null** or is a reference to an object
    - Note: the value of a variable of a primitive type is a value of that type (**false**, 42, 2.718, 'a', *etc*)
  - An object has to be created explicitly, usually by calling a constructor
    - From above 2 points: declaring a variable of the object's class does NOT create an object
  - More than one variable can refer to the same object (aliases)
    - An assignment copies the *reference* and *not* the object

Getters and Setters

- In Student, as `name` and the three test scores are **public** and not **final**, other classes (like Application) can change their values directly by assigning new values to them
  - These assignments change the state of the object
- In object-based and object-oriented programming it is standard practice to prevent this by making these fields **private**
- If other classes need access to the values, can provide methods that return them (these are called *getters*)
- If other classes need to change the values, can provide methods that update them (these are called *setters)*

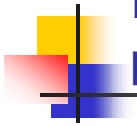


Getters and Setters continued...

- An advantage of setters is that the methods can include logic to restrict what values can be assigned
  - For the test grades a setter could ensure that the grade is always between 0 and 100, for example
- Technically, getters and setters relate to *properties* of the class
  - Properties are typically represented or captured by the values of one or more instance variables
- Conventions
  - Getters are named *getX*() where X is the name of the property (or *isX*() when the property is **boolean**)
  - Setters are named *setX()* where X is the name of the property

public versus **private**

- Any field (variable) or method that is **public** in a class is part of the *interface* of the class
  - this is the same concept as the interface or contract for a subroutine that was discussed in the lecture on subroutines
- Other classes that use the class will rely on this interface and use the **public** members
- It is very difficult to change anything in a class that is **public** as all code that uses the class needs to be changed as a result or it may no longer compile
- Things should only be made **public** if other classes need access to them in order to use your class



Revised Student class (textbook, page 204)

```
public class Student {
    private String name; // The Student's name.
    public double test1, test2, test3; // Grades on 3 tests.

    Student(String theName) {
         // Constructor for student objects;
         // provides a name for the Student.
         // The name cannot be null.
        if (theName == null)
              throw new IllegalArgumentException(
                     "name can't be null");
        name = theName;
    }
```

Revised Student class cont'd

```
    public String getName() { // Getter method for reading the
           return name;       // value of the private instance
    }                         // variable, name.

    public double getAverage() { // Compute average test grade.
        return (test1 + test2 + test3) / 3;
    }

} // end of class Student
```


Using the Student class

- Now the original code (page 194 of the book) that uses the Student class will not compile

```
public class Application {
  public static void main(String[] args) {
    Student std, std1, std2, std3;// declare Student variables
    std = new Student(); /* Does not compile, no default
                            constructor for Student class*/
    std1 = new Student();
    std2 = std1; // std2 and std1 both refer to second Student
    std3 = null; // store a null reference in std3
    std.name = "John Smith";  // Does not compile as name is
    std1.name = "Mary Jones"; // now private and not visible
  }
}
```


Using the revised class

- This version of Application achieves the same effect as the original one with the revised class:

```
public class Application {
  public static void main(String[] args) {
    Student std, std1, std2, std3;// declare Student variables
    std = new Student("John Smith"); // create first Student
    std1 = new Student("Mary Jones"); // create second Student
    std2 = std1; // std2 and std1 both refer to second Student
    std3 = null; // store a null reference in std3
  }
}
```


Comments on revised class

- Name is now **private**, is set by the constructor and cannot be changed
  - The constructor throws an IllegalArgumentException if an attempt is made to create a Student without providing a String for the Student's name
- Two getters, one for *name* and one for *average*
  - note, `average` is a property captured by the values of all three test grades

The revised class...

- The three test scores are still declared as **public**
  - It must be possible to change their values, as the Student only sits the tests after their record is created, and they are just initialized to 0.0
- One should also make the three test scores **private** and provide setters and getters for them

Adding setters to the Student class

```
private double test1, test2, test3; // scores now private

// Setter method for updating the first test score
public void setTest1(double score) {
  if (score < 0.0 || score > 100.0)
          throw new IllegalArgumentException(
                          "Score must be a percentage");
  test1 = score;
}

public double getTest1() { // Getter method for reading the
       return test1;       // value of the private instance
}                          // variable, test1.

// … getters and setters for the other two tests
```


Objects in Java

- Objects in Java are created as instances of classes
  - A class is a template for creating objects
- Every class has at least one constructor
  - Calling the constructor creates and initializes an instance of the class and returns a reference to that instance
- Information about the state of the instance is stored in the instance variables of the object
  - Normally declared as **private** to control access to them
- The instance methods can see these variables directly and do not need to be passed them as parameters by the caller
  - Instance methods are declared as **public** if it is intended that code from elsewhere in the system should be able to call them

Reading

- Today's material was based on a subset of sections 5.1 and 5.2 of the textbook
- Read these sections for next week

Questions??