Πανεπιστήμιο Κύπρου
Τμήμα Πληροφορικής

ΕΠΛ233: Αντικειμενοστρεφής Προγραμματισμός

Διδάσκων: Δρ. Χαράλαμπος Πουλλής

Εργαστηριακή Άσκηση 3

Πρώτο Μέρος: Θεωρία (διάρκεια 15 λεπτά)

Από την ιστοσελίδα της Java διαβάστε τα ακόλουθα άρθρα:

- What is an Object? http://java.sun.com/docs/books/tutorial/java/concepts/object.html
- What is a Class? http://java.sun.com/docs/books/tutorial/java/concepts/class.html
- What is Inheritance? http://java.sun.com/docs/books/tutorial/java/concepts/inheritance.html
- What Is a Package? http://java.sun.com/docs/books/tutorial/java/concepts/package.html

Απαντήστε στις ακόλουθες ερωτήσεις:

Real-world objects contain ___ and ___.

A software object's state is stored in ___.

A software object's behavior is exposed through ___.

Hiding internal data from the outside world, and accessing it only through publicly exposed methods is known as data ___.

A blueprint for a software object is called a ___.

Common behavior can be defined in a ___ and inherited into a ___ using the ___ keyword.

A collection of methods with no implementation is called an ___.

A namespace that organizes classes and interfaces by functionality is called a ___.

The term API stands for ___?

Δεύτερο Μέρος: Προγραμματισμός (διάρκεια 1 ώρα)

A class stores the attributes (data) of something.

Group values. Classes are used to group a number of related, named, data values associated with an entity. By entity we mean something that is typically a noun when you are talking about a problem you are solving. For example, in a university computing system, you might have a class to represent a student, an instructor, a classroom, a department, a course, a section of a course, ...

Student example. The information you would store about students, would be their name, id, etc. We'll keep this example simple and just save name and id information.

Declare each field. A class contains declarations of the fields (instance variables, attributes) that hold the data associated with a student. These declarations are like declaring a local variable in a method, except that you will also specify the visibility. We'll start by using the visibility modifier public, which lets anyone see them. For example, the following Student1 class can be used to represent a student. As you go through these notes, we'll make improvements to this class.

```
public class Student1 {
        public String firstName;
        public String lastName;
        public int    id;
}
```

Typically there is one class definition per file, so this would be stored in Student1.java.

Use new to create a new object: A class defines what fields an object will have when it's created. When a new Student1 object is created, a block of memory is allocated, which is big enough to hold these three fields -- as well as some extra overhead that all objects have.

```
Student1 tatiana;
tatiana = new Student1();   // Create Student1 object with new.
```

"new" and parentheses: To create a new object, write new followed by the name of the class (eg, Student1), followed by parentheses. Later we'll see that we can specify arguments in the parentheses when creating a new object.

Default field values - null, zero, false: Unlike local variables in a method, fields do have default values (like the default values in arrays). Object references are null, numbers are zero, and booleans are false.

Access public fields with dot notation: The fields (firstName, lastName, id) name data which is stored in each object. Another term for field is instance variable. All public fields can be referenced using dot notation (later we'll see better ways to access and set fields). To reference a field, write the object name, then a dot, then the field name.

```
public class TestStudent1      {
        public static void main(String [] args)        {
                Student1 tatiana;               // Declare a variable to hold a Student1 object.

                //... Create a new Student1 object for Tatiana.
                tatiana  = new Student1();        // Create a new Student1 object with default values.
                tatiana.firstName = "Tatiana";    // Set values of the fields.
                tatiana.lastName  = "Johnson";
                tatiana.id        = 9950842;
        }
}
```

A class definition is a template for creating objects: A class defines which fields an object has in it. You need to use new to create a new object from the class by allocating memory and assigning a default value to each field. A little later you will learn how to write a constructor to control the initialization.
It's not very interesting to create only one object of class, so the following example creates a couple of them.

```
// File   : TestStudent1.java
import javax.swing.*;/*Swing is a widget toolkit for Java. It is part of Sun Microsystems' Java Foundation
                       Classes (JFC) — an API for providing a graphical user interface (GUI). */

public class TestStudent1 {
   public static void main(String[] args) {
      Student1 tatiana;
      Student1 pupil;

      //... Create new Student1 object with new.
      tatiana = new Student1();
      tatiana.firstName = "Tatiana";
      tatiana.lastName  = "Johnson";
      tatiana.id        = 9950842;

      //... Create another Student1 object.
      pupil = new Student1();
      pupil.firstName = JOptionPane.showInputDialog(null, "First name");
      pupil.lastName  = JOptionPane.showInputDialog(null, "Last name");
      pupil.id        = Integer.parseInt(JOptionPane.showInputDialog(null, "ID"));

      JOptionPane.showMessageDialog(null, "One student is named: "
          + tatiana.lastName + ", " + tatiana.firstName
          + "\n and another is named: "
          + pupil.lastName + ", " + pupil.firstName);
   }
}
```

Add a constructor for better initialization: Avoid bad initializations. One problem with the Student1 class is that the user has to explicitly initialize all fields. This requires extra typing, but more importantly, it is error-prone. If we forget to initialize a field, the default value could have bad consequences.
Convenience. Defining a constructor makes creation of an object easier to write.

Constructor Syntax: Similar to method. A constructor is similar to a method -- it's called like a method, has parameters like a method, and it returns. But it must have the same name as the class for which it is a constructor. Also, the type and return value are implicit.

Student2 example with a constructor: Here is the Student1 class with a constructor which simply initializes the fields (instance variables) from its parameters. Often constructors do more work in initialization, including checking for legal values, but for now we'll just simply copy the parameter values.

```
// File   : Student2.java
// Purpose: Information about a student. Defines constructor.

public class Student2 {
   public String firstName; // First name
   public String lastName;  // Last name
   public int    id;        // Student id

   //====================================== constructor
   public Student2(String fn, String ln, int idnum) {
      firstName = fn;
      lastName  = ln;
      id        = idnum;
   }
}
```

Note that a constructor has no return type and the name is the same as the class name.

Using the constructor: Here is the first program rewritten to use the above class definition with a constructor.

```
// File   : TestStudent2.java
// Purpose: Tests Student2 constructor.

import javax.swing.*;

public class TestStudent2 {
   public static void main(String[] args) {
      Student2 tatiana;
      Student2 pupil;

      //... Create new Student2 object with new.
      tatiana = new Student2("Tatiana", "Johnson", 9950842);

      //... Create another Student2 object.
      String first = JOptionPane.showInputDialog(null, "First name");
      String last  = JOptionPane.showInputDialog(null, "Last name");
      int    studID= Integer.parseInt(JOptionPane.showInputDialog(null, "ID"));
      pupil = new Student2(first, last, studID);

      JOptionPane.showMessageDialog(null, "One student is named: "
          + tatiana.lastName + ", " + tatiana.firstName
          + "\n and another is named: "
          + pupil.lastName + ", " + pupil.firstName);
   }
}
```

When you define a constructor, the Java compiler doesn't create a default constructor: If you define a constructor in a class, the Java compiler no longer automatically creates a default (parameterless) constructor. All object creation therefore must use your explicitly defined constructor. For example,

```
Student2 someone;
someone = new Student2();          // ILLEGAL.  There is no default constructor.
someone = new Student2("Michael", "Maus", 1);  // OK.  Must specify 3 values.
```

The constructor can check that all fields are defined with legal values. We're not going to extend the Student2 class any further to test for legal values, but we will in the next example.

For the moment we'll leave the Student class, and move to something different to show the same ideas.

TimeOfDay1 Example - No constructor: Let's say that we want to represent a time of day necessary for representing when an appointment is scheduled. It won't represent the day, only the time as hour and minutes. Here's a start using a 24 hour clock (00:00 to 23:59).

```
// File   : TimeOfDay1.java
// Purpose: A 24 hour time-of-day class.
//          This simple value class has public fields.

public class TimeOfDay1 {
   public int hour;
   public int minute;
}
```

```
// File   : TimeTest1.java
// Purpose: Test the TimeOfDay1 class.
//          Shows use of dot notation to select public fields.

import javax.swing.*;

public class TimeTest1 {
   public static void main(String[] args) {

      //... Create a new TimeOfDay object.
      TimeOfDay1 now = new TimeOfDay1();

      //... Set the fields.
      now.hour   = 10;  // Set fields explicitly.
      now.minute = 32;

      //... Display the time by referencing the fields explicitly.
      JOptionPane.showMessageDialog(null, now.hour + ":" + now.minute);
   }
}
```

TimeOfDay1b.java - As above, but with constructor:

```
// File   : TimeOfDay1b.java
// Purpose: A time-of-day class with constructor, which provides
//          a more convenient and conventional way to build objects.
//          The constructor is overloaded (more than one version) for
//          more convenience.

public class TimeOfDay1b {
   public int hour;
   public int minute;

   //=================================================== constructor    //Note 1
   public TimeOfDay1b(int h, int m) {
      hour   = h;   // Set the fields from the parameters.
      minute = m;
   }
}
```

Notes: Constructors have no return type, and have the same name as the class.

Test program using constructor:

```
// File   : TimeTest1b.java
// Purpose: Test the TimeOfDay1b class.

import javax.swing.*;

public class TimeTest1b {

   public static void main(String[] args) {

      //... Create a TimeOfDay1b object for 10:30 in the morning.
      TimeOfDay1b now = new TimeOfDay1b(10,30);
```

```
      //... Display it.
      JOptionPane.showMessageDialog(null, now.hour + ":" + now.minute);
   }
}
```

Overloading constructors: It's common to overload constructors - define multiple constructors which differ in number and/or types of parameters. For example, exact hours are common, so an additional constructor could be defined which takes only the hour parameter. You can then set to minutes to a default value.

```
// File   : TimeOfDay1c.java
// Purpose: A time-of-day class with constructor, which provides
//          a more convenient and conventional way to build objects.
//          The constructor is overloaded (more than one version) for
//          more convenience.

public class TimeOfDay1c {
   public int hour;
   public int minute;

   //================================================== constructor
   public TimeOfDay1c(int h, int m) {
      hour   = h;
      minute = m;
   }

   //================================================== constructor
   public TimeOfDay1c(int h) {
      hour   = h;
      minute = 0;   // Set minutes to 0.
   }
}
```

Calling one constructor from another using "this(...)":
Default parameters. It's very common for a constructor with fewer parameters to call a constructor with more parameters, supplying default values. For this usage, the constructors with fewer parameters will frequently consist of only the "this" call.

"this". Instead of calling the constructor with the class name, use the keyword this. The compiler matches "this" with a constructor with the appropriate number and types of parameters, and calls it.

Must be first. The "this" call must be the very first line of the constructor.

```
// File   : TimeOfDay1d.java
// Purpose: A time-of-day class with overloaded constructors.
//          One constructor calls the other using "this".

public class TimeOfDay1d {
   public int hour;
   public int minute;

   //================================================== constructor
   public TimeOfDay1d(int h, int m) {
      hour   = h;
      minute = m;
   }
```

```
    //================================================== constructor
    public TimeOfDay1d(int h) {
        this(h, 0);   // Call other constructor.
    }
}
```

Hidden constructor call at the beginning of every constructor: The one essential way in which constructors differ from methods is that the first statement of every constructor is either a call on the constructor for the superclass (using super) or a call to another constructor in the same class (using this).

Hidden call. You normally don't see the super call because the compiler automatically generates a call to the parameterless superclass constructor for you, but it's generated at the beginning of every constructor which doesn't explicitly call another constructor.

```
public TimeOfDay1d(int h, int m) {
    hour   = h;
    minute = m;
}
```

Is the same as:

```
public TimeOfDay1d(int h, int m) {
    super();   // Call Object constructor.
    hour   = h;
    minute = m;
}
```

"Copy" Constructors: Another reason to overload constructors is to provide a copy constructor, which builds one object from the values of another similar object.

```
// File   : TimeOfDay1d.java
// Purpose: A time-of-day class with overloaded constructors.
//          Implements a copy constructor.

public class TimeOfDay1e {
    public int hour;
    public int minute;

    //================================================== constructor
    public TimeOfDay1e(int h, int m) {
        hour   = h;
        minute = m;
    }

    //================================================== constructor
    public TimeOfDay1e(int h) {
        this(h, 0);   // Call other constructor.
    }

    //================================================== constructor
    public TimeOfDay1e(TimeOfDay1e other) {
        this(other.hour, other.minute);   // Call other constructor.
    }
}
```

Questions: What would happen if you called the following constructor?

```
TimeOfDay1c dentistAppointment = new TimeOfDay1c();
```

Πρόγραμμα: Προβολή Mercator. Η προβολή Mercator (Mercator projection) είναι μια σύμμορφη (διατηρεί τις γωνίες) προβολή που μετατρέπει το γεωγραφικό πλάτος (latitude) φ και γεωγραφικό μήκος (longitude) λ σε καρτεσιανές συντεταγμένες (x, y). Είναι ευρέως χρησιμοποιούμενη – για παράδειγμα, σε ναυτικούς χάρτες και στους χάρτες που τυπώνεται από το διαδίκτυο. Η προβολή ορίζεται από τις ακόλουθες εξισώσεις:

$$x = \lambda - \lambda 0$$
$$y = 1/2 \ln((1 + \sin \varphi) / (1 - \sin \varphi))$$

όπου λ0 είναι το γεωγραφικό μήκος του σημείου στο κέντρο του χάρτη. Γράψτε ένα πρόγραμμα όπου παίρνει το λ0, το γεωγραφικό πλάτος (latitude) φ και γεωγραφικό μήκος (longitude) λ από τη γραμμή εντολών και τυπώνει την προβολή τους. Το όνομα του προγράμματος σας πρέπει να είναι Mercator.