Java

Enhancing classes

Experience is something you don't get till just after you need it

Lecture objectives

- We can now explore various aspects of classes and objects including analysis and design

- You should be able to understand
  - overloading constuctors
  - an overview of object-oriented analysis
  - `this` reference
  - the static modifier
  - Arrays of objects

Review Instantiable classes

Ensure you are totally at ease with:

- Writing instantiable classes
  - Private instance variables
  - Constructors
  - Instance methods

- Writing a driver program
  - Creating objects from instantiable classes
  - Using the `new` keyword to create objects
  - Invoking instance methods

```
import Account;  // unnecessary if located in the same folder
public class TestAccount
{
   public static void main (String[] args)
   {
      Account acct1 = new Account (“J Bond", 72354, 102.56);
      Account acct2 = new Account (“M Munro", 69713, 40.00);

      acct1.deposit (25.85);
      acct2.deposit (500.00);
      double currentBal = acct1.getBalance();
      System.out.println (“acct1 balance: " +   currentBal);

      acct2.addInterest();
      currentBal = acct2.getBalance();
      System.out.println (“acct2 balance: " +   currentBal);
    }
}
```

```
//  Account.java
public class Account
{
   private final double RATE = 0.045;  // interest rate 4.5%
   private int acctNumber;
   private double balance;
   private String name;

   public Account (String owner, int account, double initial)
   {
      name = owner;
      acctNumber = account;
      balance = initial;
   }
   public void deposit (double amount)
   {
       balance = balance + amount;
    }

  public double getBalance()
   {
      return balance;
   }

   public void addInterest ()
   {
     balance = balance + (balance * RATE);
   }
}
```

Exercise

- Construct a Circle class definition
  - Instance variables: radius, area
  - Constructor: takes one integer parameter for radius
    - initialise radius to parameter value
    - initialise area to 0
  - Methods:
    - setRadius (takes an integer parameter)
    - calculateArea (no parameters, no return; calculates area)
    - getArea (returns area)

- Construct Driver program
  - Prompt user for circle radius
  - Create a Circle object (passing radius as parameter)
  - Calculate its area
  - Get area and display
  - Set radius to 10, get area and display

Overloading Constructors

- Recall method overloading?
- Like methods, constructors can be overloaded
- An overloaded constructor provides multiple ways to set up a new object

```
Die die1 = new Die();          //creates a 6 sided die
Die die2 = new Die(20);        //creates a 20 sided die
```

```
//  SnakeEyes.java       Author: Lewis and Loftus
import Die; //no need if Die.java is in the same folder
public class SnakeEyes
{
public static void main (String[] args)
  {
     final int ROLLS = 500;
     int snakeEyes = 0, num1, num2;

     Die die1 = new Die();      // 6 sided die
     Die die2 = new Die(20);  // 20 sided die


     for (int roll = 1; roll <= ROLLS; roll++)
     {
        num1 = die1.roll();
        num2 = die2.roll();

        if (num1 == 1 && num2 == 1)  // snakes eyes
           snakeEyes =snakeEyes + 1;
     }
      System.out.println ("Ratio: " +
                          (double)snakeEyes/ROLLS);
  }
}
```

```
//  Die.java       Author: Lewis and Loftus
public class Die
{
     private int numFaces;       // sides on the die
     private int faceValue;       // current face

public Die ()       //constructor
{
     numFaces = 6;
     faceValue = 1;
 }

public Die (int faces)   //overloaded constructor
 {
       numFaces = faces;
       faceValue = 1;
 }


public int roll()
{
     faceValue = (int)(Math.random()
                                   * numFaces) + 1;
     return faceValue;
}
```

Object-Oriented Analysis

- One of the simplest ways to do object-oriented analysis involves finding nouns and verbs in the program's specification.
- Many of the nouns will represent *classes,* and many of the verbs will represent *methods*.
- This simple technique doesn't work perfectly, but it can be useful for generating ideas.

Object-Oriented Analysis

- So an initial analysis of a banking system might reveal:
- Nouns ie suggested classes in the specification for a banking system:
  - Customer
  - Account
- Verbs ie suggested methods in the specification for a banking system:
  - Open (an account)
  - Deposit (money into an account)
  - Withdraw (money from an account)
  - Close (an account)

Categories of Methods

- Most instance methods fall into one of several categories:
  - *Manager methods*
  - *Implementor methods*
  - *Access methods*

Manager Methods

- Initialise objects
  - Constructors
  - Having additional constructors makes a class more useful.

Implementor Methods

- *Implementor* methods represent useful operations that can be performed on instances of the class.
- *Implementor* methods usually change the state of an object
- The `Account` class *implementor* methods:
  - `open`,
  - `deposit`,
  - `withdraw`,
  - `close`.

Access Methods

- For access to instance variables, the choice is:
  - provide instance methods or
  - make the variables public (but this violates good object orientation)

- *Accessors* (or *getters*)
  - methods that *return* the value of a private variable.
  - by convention, names of *getters* start with the word `get`
- *Mutators* (or *setters*)
  - methods that *change* the value of a private variable.
  - by convention, names of *setters* start with the word `set`

The `this` Reference

- Because an instance method is called by an object, the correct instance data is accessed

```
double balance  = acct1.getBalance();
```

- The keyword `this` is used to refer to the object that invoked the method
- In the above, the `this` reference refers to `acct1`

The `this` Reference

- The following two methods are actually identical. The first is using the `this` reference implicitly, the second explicitly

```
public double getBalance ()
{
   return balance;
}
```

```
public double getBalance ()
{
   return this.balance;
}
```

Variable Type Differences

Local variables and parameter variables

- Are declared within a method
- Only exist for the duration of the method's execution
- Need to be initialised
  - A method is not allowed to access the value stored in a local variable until the variable has been initialised

Instance variables

- belong to an object
  - are created when an object is constructed
  - exist as long as the object exists
- each object has its own copy
  - that's what allows objects to be different
- are automatically initialised (if not explicitly set in a constructor),
  - `zero` for numeric and char variables
  - `false` for boolean variables
  - `null` for reference type variables
- every instance variable should be initialised in a constructor

Hidden Variables

- We cant declare a variable with the same name as one already in scope
  - always use different variable names except in headers of *for loops*

- Beware of variable names with instantiable classes!!!
- An instance variable and a local variable could have the same name –
  - the local variable with the inner scope will take precedence over (or hide) the variable in the outer scope (instance variable)

Recall the Account class

```
public class Account
{
        private int acctNumber;
        private double balance;
        private String name;

        public Account (String owner, int account, double initial)
        {
                acctNumber = account;
                balance = initial;
                name = owner;
        }
        …..
```

Hidden variable

- What if we named the parameter variable `balance`

```
private int acctNumber;
private double balance;
private String name;


public Account(String owner, long account, double balance)
{
    balance = balance;
    ......
}
```

- The local variable will overwrite itself
- The `balance` instance variable is hidden

`this` used in Constructors

- The `this` reference can be used to refer to the instance variable of the object

```
private long acctNumber;
private double balance;
private String name;

public Account(String owner, long account, double balance)
{
    this.balance = balance;    // and so on for the other variables
}
```

- This approach is eliminates the need for different yet equivalent names
- It is often found in constructors

The `static` Modifier

- We have used methods that do not require objects
  - eg methods of `Math` class
- These *static* or *class methods* are invoked through the class name - not through an object name
- To make a method static, we apply the `static` modifier to the method definition

```
static double sqrt (double num)
```

- And call the method by the class name

```
System.out.print ("sq rt of 9: " + Math.sqrt(9));
```

Static (or Class) Variables

- The `static` modifier can be applied to variables as well
- It associates a variable with the class rather than an object
- They are declared with the word `static`
- Normally, each object has its own data space
- If a variable is declared as `static`, only one copy of the variable exists

- Eg in the Account class a useful class variable may be

```
private static double totalDeposits;
```

Static Variables

- All objects created from the class share access to the static variable
- Changing the value of a static variable in one object changes it for all others
- Eg to allocate a unique asset number every time a new asset object is created
  - A static variable could hold the last used asset number
  - Any new asset could increment this number by 1

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

      Song obj1 = new Song (“Friends”);
      System.out.println (obj1.getSongTitle());

      Song obj2 = new Song ("Don't Worry. Be Happy");
      System.out.println (obj2.getSongTitle());

      Song obj3 = new Song (“She’s a bop girl");
      System.out.println (obj3.getSongTitle());

       System.out.println();

      System.out.println ("Songs created: " +
                                          obj2.getCount());
   }
}
```

```
Friends
Don't Worry. Be Happy
She’s a bop girl

Songs created: 3
```

```
public class Song
{
    private String songTitle;
    private static int count = 0;
    //  Sets up slogan and counts the number of instances

    public Song (String str)
    {
       songTitle = str;
      count = count + 1;
    }

    //  Returns this slogan as a string.
    public String getSongTitle()
    {
      return songTitle;
    }

    //  Returns the number of instances of this class
    public int getCount ()
    {
      return count;
    }
}
```

Arrays of objects

- Because each object has a unique identifier we cant use them in loops as we would do with variables

```
Account acct1 = new Account((“J Bond", 72354, 102.56);
acct1.deposit(100.00);
```

- By storing objects in an array we can use the power of loops to process multiple objects

Arrays of objects

- Create an array of objects

```
Account[ ] acctArray = new Account[3];
```

- No actual objects have been created at this time
  - This is merely an array waiting to receive Account objects
- We need to create each object in the usual way
  - Call the constructor of the Account class
  - Pass it any parameters it requires
  - Assign it to a position in the array

```
acctArray[index] = new acctArray(name, number, balance);
```

Arrays of objects

- After you have instantiated the array with actual objects
  - You can use getters and setters or other methods from the Account class to return or change instance variables

```
balance = acctArray[index].getBalance();
                    or
System.out.println(acctArray[index].getBalance());
```

Using an array as a database

- Use a *for* loop to search every position in an array

```
int index;
for (index = 0; index < acctArray.length; index ++)
{
      if (acctArray[index].getAcctNumber == 72354)
          break;
}
```

- When the loop has terminated
  - Test whether *index* is less than *acctArray.length*
  - if so *index* indicates the position of the record in the array
  - Use index to modify that object

```
acctArray[index].deposit(amount);
```

Comparing Strings for equality

- Remember to use the *equals()* method not ==

```
if (acctArray[index].getName() .equals (“J Bond”)
```

```
String numStr = JOptionPane.showInputDialog ("Enter account number ");
int num = Integer.parseInt(numStr);

int i;
for (i = 0; i < acctArray.length; i++)
{
    if (acctArray[i].getAcctNumber() == num)
       break;
}

if (i < acctArray.length)
{
   JOptionPane.showMessageDialog
                     (null, "The existing name is " + acctArray [i].getName());
   String newName = JOptionPane.showInputDialog
                     ("Enter new name ");
   acctArray [i].setName(newName);
}
```

Lecture Outcomes

Today we have covered:

- an overview of object-oriented analysis
- `this` reference
- the static modifier
- Arrays of objects

- Questions?