ITEC 1630 Winter 2007
Week 1: Review of Object-Based Programming Part 2
By Yves Lesperance
(these notes are adapted from those of Radu Campeanu)

Choosing Classes

- objects correspond to nouns in the description of your application
- their behavior corresponds to *public* methods contained in their corresponding classes
- choosing classes is not a trivial task
- occasionally classes are not built to generate objects but to collect static methods and constants. Ex: Math, Color, Integer
- create classes which represent a single concept

E.g. In class Purse shown below there are 2 concepts: a purse that holds coins and the value of each coin.

```
class Purse {
   public Purse();
   public void addNickels(int count);
   public addDimes(int count);
   public addQuarters(int count);
   public double getTotal();
}
```

It is therefore better to rewrite the class as:

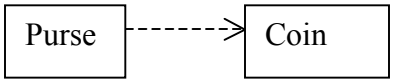
```
class Coin{
   public Coin(double value, String aName);
   public double getValue();
}
class Purse {
  public Purse();
  public void add(Coin aCoin);
  public double getTotal();
}
```

In this case class Purse *depends* on class Coin as it is using objects of Coin type. Of course, class Coin does not depend on class Purse.

UML class diagrams

The world of classes and objects is conveniently presented in the Unified Modeling Language (UML) diagrams. See Horstman ch 9 and Appendix N.

In UML the relationship between Purse and Coin, that Purse **uses** Coin, is represented with an arrow pointing to Coin, the dependent class:



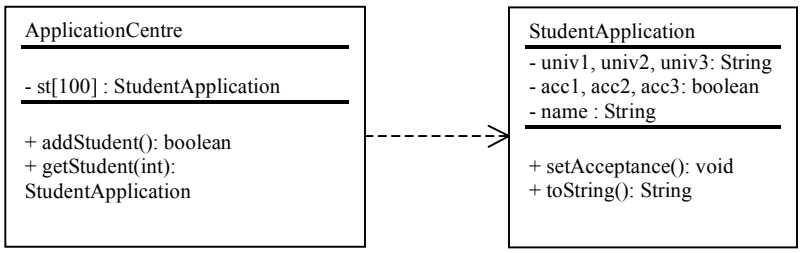
If class Coin changes that will imply that class Purse has to be rewritten. In general you want to minimize the number of class dependencies (low coupling).

A More Complex Example: A University Applications Centre

In the remaining part of this Review of ITEC1620 let us consider an Application Centre for universities admissions. A student can choose 3 different universities and let us assume that this center will accept up to 100 different applications. This program has to allow the user to:

i) enter the student application,
ii) accept students in some of the universities that they chose
iii) display the status of the applications.

The two obvious classes are the ApplicationCentre and the Student. The Student class is the dependent class but unlike in the previous example the relationship between the two classes is of the type one-to-many. The following *UML **class** diagram* will specify the relation between Student and ApplicationCentre.



Note: A *UML **object** diagram* contains a snapshot of the objects at a certain point in the execution of the program. The notation will include the names of the objects and the values of the attributes. We shall note use UML object diagrams in this course.

Java Implementation of the Applications Centre

The Applications class presented below is the driver which allows for the *association* of ApplicationCentre and StudentApplication classes. For input/output we shall use JOptionPane, a class part of the Swing package which produces simple pop-up dialog boxes; see Horstmann p. 139 and 1114. (In most of this course we shall avoid the line input/output used in ITEC1620.)

```
// A class which handles applications for university admission
import javax.swing.JOptionPane;

public class Applications {
   public static void main( String args[] ) {
       ApplicationCentre appCentre= new ApplicationCentre ("Guelph");
       String stopper="quit";
                int nrStud=0;

// Input data

            JOptionPane.showMessageDialog(null,"Enter student names, each
followed by 3 choices of universities. To stop enter for name the word
quit","Input", JOptionPane.PLAIN_MESSAGE);
       String n = JOptionPane.showInputDialog("Student's name ?");
       boolean flag=true;
       while (!n.equals(stopper) && flag){
             String u1= JOptionPane.showInputDialog("1st university ?");
             String u2= JOptionPane.showInputDialog("2nd university ?");
                              String u3= JOptionPane.showInputDialog("3rd
university ? ");
             StudentApplication s=new StudentApplication (n, u1, u2, u3);
             flag=appCentre.addStudent(s);
                              n=JOptionPane.showInputDialog("Student's name ?");
                              nrStud++;
       }

  // Accept some applications

       String line = JOptionPane.showInputDialog(" Indicate which applications
are to be accepted by entering the student index number. To stop enter for name
the word quit");
       while (!line.equals(stopper)) {
         int stnr=Integer.parseInt(line);
         line= JOptionPane.showInputDialog("Enter university index 0..2");
         int anr=Integer.parseInt(line);
         appCentre.getStudent(stnr).setAcceptance(anr, true);
                 line= JOptionPane.showInputDialog(" Indicate which applications
are to be accepted by entering the student index number. To stop enter for name
the word quit");
               }

  // Print the final results
                   JOptionPane.showMessageDialog(null, " Records for all
applicants to "+ appCentre.getName() + "\n","Records",
JOptionPane.PLAIN_MESSAGE);
             for (int i=0;i<nrStud; i++)

JOptionPane.showMessageDialog(null,appCentre.getStudent(i).toString()
,"Records", JOptionPane.PLAIN_MESSAGE);
   }
System.exit(0);
}
```

```
class ApplicationCentre {
  private String name;
  private StudentApplication [] st;
  private int studentCount;
  private int size;

  public ApplicationCentre(String s){
        name=s;
        size=100;
        st = new StudentApplication[size];
        studentCount=0;
}

  public String getName() {
        return name;
  }

  public boolean addStudent(StudentApplication s){
   if (studentCount==size) return false;
   st[studentCount]=s;
   studentCount ++;
   return true;
  }

  public StudentApplication getStudent(int which){
      if (which<0 || which > studentCount-1){
         return null;
      }
        return st[which];
  }
}

  class StudentApplication{
      private String name;
      private String university0;
      private String university1;
      private String university2;
      private boolean accept0;
      private boolean accept1;
      private boolean accept2;

      public StudentApplication (String n, String u0, String u1, String u2){
            name = n;
            university0=u0;
            university1=u1;
            university2=u2;
            accept0=accept1=accept2=false;
      }

      public void setAcceptance(int which, boolean decision){
            switch(which){
            case 0: accept0=decision; break;
            case 1: accept1=decision; break;
            case 2: accept2=decision; break;
            }
      }
```

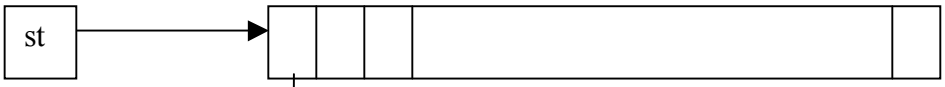
```
    public String toString(){
          String result = name + ":\n";
                           result += university0;
          if (accept0) result += " - accepted\n";
                else result += " - rejected\n";
                   result += university1;
          if (accept1) result += " - accepted\n";
                else result += " - rejected\n";
                           result += university2;
          if (accept2) result += " - accepted\n";
                else result += " - rejected\n";
                           return result;
    }
}
```

The most important concept in this program is the creation and use of the array of StudentApplication objects st[].

The lines

```
size=100;
st = new StudentApplication[size];
```

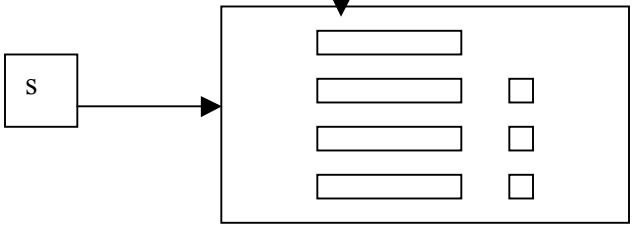
in the ApplicationCentre constructor creates an array of null references:



The lines

```
StudentApplication s = new StudentApplication (n, u1, u2, u3);
flag = appCentre.addStudent(s);
```

in the main() of class Applications create the object s of StudentApplication type and then through the method addStudent() of class ApplicationCentre the address of s is passed to an element of the st[ ] array.



This operation is repeated in the while loop until either the array is full or the user enters the word “quit” as the name of the student.