# Hanout-17
# Introduction to Abstraction

## Overview

- Introducing the concept of abstraction
- Functional abstraction
- Functional abstraction in action
- Practice exercises
- Preview: Methods

# Introduction to Abstraction

Consider the following program which aims to input and output the average quiz scores for two students

```
import java.io.*;
import TextIO;
class IntroToAbstraction {
  static TextIO inputStream = new
  TextIO(System.in);
  public static void main (String args[])
  throws IOException {
    int quiz1, quiz2, quiz3, sum;
    System.out.println("Enter your (expected)
  scores for 3 quizzes :");
    quiz1 = inputStream.readInt();
    quiz2 = inputStream.readInt();
    quiz3 = inputStream.readInt();
    sum = quiz1 + quiz2 + quiz3;
    System.out.println("Your (expected)
  average score is :" + sum/3.0);
    System.out.println("Enter the (expected)
  scores for your friend:");
    quiz1 = inputStream.readInt();
    quiz2 = inputStream.readInt();
    quiz3 = inputStream.readInt();
    sum = quiz1 + quiz2 + quiz3;
    System.out.println("Your friend's
  (expected) average score is :" + sum/3.0);
  }
}
```

# Introduction to Abstraction (cont.)

- You're asked to improve the above program to cater for all students in your class, not just for you and your friend.
- You'll notice that modifying the program along the lines it is written will become unbearably long and hard to read.
- How do we improve the program in such a way that it does not become unnecessarily longer and not much more difficult to read than the original program?
- The answer to this is based on what is generally called as *abstraction* which we now introduce before modifying the program.
- Abstraction refers to the process of simplification by *hiding or eliminating unnecessary detail*.

# Introduction to Abstraction (cont.)

- Abstraction is a powerful technique for describing something in terms of its essential attributes, ignoring the non-essential attributes.
- A popular form of abstraction used in virtually all programming languages is called *procedural / functional* abstraction.
- Procedural abstraction is the technique of organizing a sequence of instructions into a named procedure, *namedAction(p1,p2,…,pn).*
- The procedure *namedAction(p1,p2,…,pn)* is now a named unit of action which, subsequently, can be *called* whenever that action is desired.
- Whenever this named action is to be used, it is sufficient to know only *what* the action does, not *how* it achieves the result
- At a later time, we can go back and rewrite the procedure to substitute an improved method, which changes *how* it achieves its result, without altering *what* it computes.

# Abstraction in General: Examples

- Abstraction is a central issue in Object Oriented programming and we shall come back to it later.
- Before then, we mention some benefits of abstraction which should be clearer later:
  - » Ease of use- because we need to know simply what procedure *namedAction()* does, not how it does it in order to use this action.
  - » Ease of modification-because we can change the detail of how *namedAction()* is written, without having to make distributed changes throughout the program everywhere that *namedAction()* is used.
- Example abstractions
  - » Methods in JAVA: the implementation (at the place of invocation) is hidden.
  - » Abstract data types: the representation details are hidden
  - » Abstract syntax: the details of the concrete syntax are ignored.
  - » Abstract interpretation: details are ignored to analyze specific properties.

# Applying Abstraction to Example 1

- Having introduced the concept of abstraction, we are now ready to apply functional (or method) abstraction to rewrite the previous program.

- Notice that the main action of this program, for each student, is inputing the student's three scores and finding the average.

- Applying abstraction, let us name these common actions `inputScores()` and `computeAverage()` so that the program can be generalised as follows:

# Applying Abstraction to Example 1 (cont.)

```
import java.io.*;
import TextIO;
class IntroToAbstraction2 {

   static int numberOfStudents;
   static double sum, average;
   static TextIO inputStream = new
  TextIO(System.in);
   static double quiz1, quiz2, quiz3;

  public static double computeAverage(){
      sum = quiz1+quiz2+quiz3;
            return sum/3;
  }
```

# Applying Abstraction to Example 1 (cont.)

```
public static void inputScores () throws
  IOException {
        System.out.println("Enter three
  quiz scores: ");
        quiz1 = inputStream.readFloat();
        quiz2 = inputStream.readFloat();
        quiz3 = inputStream.readFloat();
  }

  public static void main (String args[])
  throws IOException {
    System.out.println("How many students
  are in the class?");
    numberOfStudents =
  inputStream.readInt();

    for(int i =1; i<=numberOfStudents; i++)
  {
        inputScores();
        average = computeAverage();
        System.out.println("Average score
  for Student " + i
              + " is " + average);
    }
  }
}
```

# Applying Abstraction: Exercises

1. Write a method named `sumToN()` that takes an integer N as parameter and returns the sum of the integers from 1 to N.
2. Write a method named `digitsInReverse()` that takes an integer N as parameter and returns the individual digits in the number in reverse order. Display each of these digits on separate lines. For example given the input 1085 the following should be displayed:

   5

   8

   0

   1
3. Write a method named `repeat()` that accepts two parameters: one is a character, the other is an integer signifying the number of times the character is to be displayed. Incorporate this method into a class and test it.

# Applying Abstraction: Exercises

4. Write a method named `unsignedIntToBinary()` that takes an unsigned integer N as parameter and returns a binary equivalent of N.

5. The numerical method of finding the square root of a number uses the iteration formula

$$X2=(N/X1 + X1)/2$$

Where N is the number whose square root is required and X2 is a better approximation than X1. Write a program to use this technique always assuming a first approximation of 1.

Your program should repeat application of the formula until 4 decimal places of accuracy have been achieved.