Java Basics Review

Software Engineering
CITS1220
Learning to program is not easy!
Du Boulay, 1989

Need to master all these different types of difficulty at once:

- general **orientation**, what programs are for and what can be done with them
- the **notional machine**, a model of the computer as it relates to executing programs
- **notation**, the syntax and semantics of a particular programming language
- **structures**, schemas/plans for solving a problem
- **pragmatics**, the skills of planning, developing, testing, debugging and so on
Learning Styles  (Perkins et al 89)

- **Stoppers**, when confronted with a problem or lack of clear direction to proceed, stoppers simply stop and abandon all hope of solving the problem on their own.

- **Movers** are students who keep trying, experimenting, modifying their code. Movers can use feedback about errors effectively, and have the potential to solve the current problem and progress.

- **Tinkerers** are extreme movers who are not able to trace/track their program, make changes more or less at random and have little effective chance of progressing.
Classes

BankAccount

owner : String
balance : Dollars

deposit ( amount : Dollars)
withdraw ( amount : Dollars)
isoverdrawn () : boolean
A class:

- Is a **unit of abstraction** in an object oriented (OO) program
- Represents similar objects: its **instances**
- Is a kind of **software module**
  - Specifies the *structure* (properties) its instances
  - Instance variables are the “memory” of an object: its state
  - Contains *methods* to implement their behaviour, changing the object’s state
Class Diagrams (in UML)

- A class is simply represented as a box with the name of the class inside

- A class diagram may also show
  - Attributes (instance and class variables)
  - Operations (methods)
Naming classes

- Use *capital* letters
  - e.g. BankAccount not bankAccount

- Use *singular* nouns

- Use the right level of generality
  - e.g. Municipality not City

- Make sure the name has only *one* meaning
  - e.g. Bus has several meanings
Java Basics (Review)

- For more details see Java 1200 lecture notes and labs
Types

- Every variable has a *type* and can hold or refer to values only of that type
- Eight primitive types stored by *value*
  - `boolean, byte, short, int, long, char, float, double`
- Everything else is reference type
  - Library classes and user-defined classes
  - All stored by *reference*
Type Conversion

- The Java compiler does extensive type-checking to make sure that the program does not have any *obvious* problems.
- The compiler will not allow the *wrong type* of value to be assigned to a variable.

```
int x;
x = true; // cannot assign boolean to int
```
- Similarly, the compiler will not allow the wrong type of value to be used as a method or constructor argument.
Type Casting

- But what about this?
  
  ```
  int x = 20;
  double y;
  y = x;
  ```

- Although the types don’t match perfectly, the compiler knows that there is a value of type double corresponding to any value of type int. In this case the value 20.0 is assigned to y

- To change back to an int we use type casting:
  
  ```
  x = x + (int) x*0.1;
  ```

  Now x has value 22
Variables

- All variables must be declared before use and can be initialized at the same time
  
  ```java
  int x = 10;
  boolean isDone = false;
  ```

- Variables of reference type will contain the special value `null` when not referring to any object
  
  ```java
  java.awt.Color col;
  MobilePhone mp;  // user-defined class
  ```
Creating objects

- Objects are created with the keyword `new`
  ```java
  Color col = new Color(255,0,128);
  ```
- This causes a special method called the *constructor* to be run; the newly created object is stored on the heap and the reference to it is returned
  ```java
  mp = new MobilePhone("041838133",200);
  ```
Calling methods

- Each *class* defines a number of methods - these can be viewed as the things that objects of this class can *do*.

- These methods are called by specifying:
  - The target object
  - The name of the method
  - The arguments to the method

```java
mp.topUp(50);
```
Source code structure

- Source code has the following format

```java
class ClassName {
    // variables (fields)
    // constructors
    // methods
}
```
Source code - variables

- All *instance* and *class* variables are defined and optionally initialized

```java
class BankAccount {
    private int balance = 0;
    static double interestRate = 0.05;
}
```

- Instance variables belong *individually* to each object - class variables are *shared*
Source code - methods

Methods must specify in order

- An (optional) access modifier
- A return type (may be void)
- A method name
- Parameters
- The code to be run

```java
public void credit (int amount) {
    balance = balance + amount;
}
```
Constructors are special methods

- No explicit return type
- Same name as the class

```java
public BankAccount(int initBalance) {
    balance = initBalance;
}
```

Constructors “set up” objects so they are ready for further actions
Object Lifecycle

- **Birth**
  - Constructor run once when `new` is called
  - Instance variables created

- **During its existence**
  - Receives sequence of method calls which may use/alter the instance variables

- **Death**
  - When all references to the object go out of scope the object is marked for garbage collection
Class methods

- Class methods (aka static methods) are used if method does not refer to an individual object
- Can be called even if no objects exist via `ClassName.methodName(...)`
- The main method is used to *start* programs
  ```java
  public static void main(String [] args)
  ```
Access control modifiers

- Apply to methods and variables
  - public
    Any class can access
  - protected
    Only code in the package or subclasses can access
  - (blank)
    Only code in the package can access.
  - private
    Only code written in the class can access
Switch statement

class SwitchDemo {
    public static void main(String[] args) {
        int month = 8;
        switch (month) {
            case 1: System.out.println("January"); break;
                //TODO cases 2 to 7 go here
            case 8: System.out.println("August"); break;
                //TODO cases 9 to 11 go here
            case 12: System.out.println("December"); break;
            default: System.out.println("Invalid month."); break;
        }
    }
}
switch (month) {
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12:
        numDays = 31; break;
    case 4:
    case 6:
    case 9:
    case 11:
        numDays = 30; break;
    case 2:
        if ( ((year % 4 == 0) && !(year % 100 == 0)) || (year % 400 == 0) )
            { numDays = 29; } else { numDays = 28; }
        break;
    default: System.out.println("Invalid month."); break;
}