Object-oriented Software Design and Development
CCP114N

Week 7

Classes and objects in Java

Case Study: class design from specification to public interface and implementation

Classes and objects in Java

Instance fields & methods

- Object and Class revisited
  - Object
    - a collection of data values (fields), plus methods that operate on that data
  - The type of an object is called a class
    - an object is an instance of a class
    - the class
      - defines the type of each field in an object and
      - provides the methods that operate on data contained in an instance of the class
- Instance fields & methods
  - Data fields belong to a particular object
    - each object has its own copy of each instance field
  - Methods that operate on the fields of an object
  - See e.g. Circle example

Class methods and fields

Keyword: **static**

- Class methods operate on the class itself rather than on an individual instance of the class
- Class methods are declared as static methods

  Public static void main(String args[ ]) {…}

- Fields of a class are declared as static fields
  - they are class fields (not instance's field)
  - there is only one copy of a class field which is shared by all of the instances of the class.
  - Example: a variable **numberOfObjects** to record the number of instantiated objects of a class

Java classes

Hierarchy and Scopes

- The Java class hierarchy
  - The root of all Java classes is the **java.lang.Object**
  - superclass
  - subclass
- Use the extends keyword
  - **class rectangle extends shape {.... }**
- Related keywords: super and this
  - **super.aMethod( );** ‘ to call a method of the superclass
  - **this.aMethod( );** ‘ **this** refers to the object of the current class
- Scopes (visibility levels) of the fields and methods in a class code
  - **public**
  - **private**
  - **protected**: can be viewed by the sub-class

Java class

Constructor and *Finalizer* methods

- **Class Constructor**
  - Has same name as the class name
  - Called when program instantiates an object of that class
  - Initializes instance variables of an object of the class
  - Can take arguments, but *cannot return values*
    - *What does the constructor return?*
  - Class can have more than one constructor
- **Class Finalizer**
  - Release resources to system (not just memory!)
  - Java provides method `finalize` defined in `java.lang.Object`
  - Receives no parameters
  - Returns `void`
- Orders of execution of Constructors and Finalizers
  - Constructors: downward from the root (top) of the inheritance tree
  - Finalizers: upward from the current subclass to the top of the inheritance tree
    - This is done automatically in Java unless your need to perform some cleaning work e.g. closing open files.

Interface in Java

- Interface: a Java construct to define methods without any implementation
  - Example: the *Comparable* interface
- A class can *implement* an interface
  - must provide definition for every method of the interface
- Why use **Interface**?
  - A way to enforce a design
  - A method to allow multiple inheritance
    - in Java a class may extend only one class but implement several interfaces
  - A form of polymorphism
- Java Interface vs. Abstract class
  - Abstract class contains abstract methods
    - may contain 'normal' methods and fields, but Interface may not
  - Compared to Interface
    - an abstract class can be sub-classed
- In Java a (sub) class can
  - extend only one (super) class, and
  - implements multiple interfaces

Summary

- Specifics of classes and objects in Java
  - terminology
    - static
    - interface
    - abstract classes
  - class initialization and termination
  - Class inheritance
    - The java Object class
    - Keyword : final
- Petrol Station case study
  - Class definition in terms of public interfaces
  - Separate back-end from front-end classes
  - Same public interface, different implementations
    - Use of different data structures