1 Object-Oriented Programming in Java

1.1 Background
- Java is an object-oriented programming language.
- Each java source file (.java) contains the definition of a class with the same name as the file.
- Java source files are organized into packages. Each package has its own directory.
- Classes from outside the current package can be used by using the import keyword.
- In order to run a Java program, the desired class must have a public static void main() method and must be in the CLASSPATH.
- Eclipse will handle most of the CLASSPATH and directory layout issues for you, as well as compiling and running your program.

1.2 Types
- Java has 9 built-in types: boolean, char, byte, short, int, long, float, double, void. All integer types are signed.
- Java has array version of all types created using the type[] notation.
- Java also has a very extensive built-in library, with many useful classes. These are mostly contained in the Java.lang and Java.util packages.

1.3 Object Lifecycles
- Objects are created using the new keyword.
- When an object is created, its constructor is called. This is a special method with the same name as the class and no return type.
- When an object is no longer being used, the Java garbage collector automatically cleans up the memory used by the object.

1.4 Encapsulation
- Access specifiers - public/private/protected/default

1.5 Composition
- Composition is achieved by creating fields (typically private) inside classes.
1.6 Inheritance

• There are two types of inheritance in Java: extends and implements.

• extends is full implementation inheritance (is-a). All non-private fields and methods are inherited. A class can only extend up to one other class.

• Interfaces are essentially Java classes that have no method bodies or fields. Interfaces are often named with adjectives.

• implements is interface inheritance. All interface methods are inherited, but must be defined with method bodies. A class can extend any number of interfaces.

• All classes in Java are derived from the Object class.

1.7 Method Signatures, Overloading and Overriding

• The signature of a method consists of its name, return type, and parameter list.

• A method specified with several different parameter lists is said to by overloaded. The correct method will be determined by the parameters.

• A method defined in a derived class with the same signature is said to be overridden. The deepest overridden method body will be used when the method is called.

1.8 Exceptions

• Error conditions are handled using exceptions.

• Exceptions are objects derived from the Exception class.

• Methods can throw exceptions when errors occur.

• When calling a method that may throw an exception, you must enclose the call in a try/catch block. The exception is handled by the code in the catch block.

1.9 Collections and Iterators

• Java has an extensive set of container classes used to store collections of objects.

• The three basic types of containers are Lists (to store sequences of objects), Maps (or associative arrays, used to store mappings between keys and objects), and Sets (to store unique objects).

• There are several implementations of each type of containers with different underlying structures (e.g. ArrayList, LinkedList).

• Objects are retrieved from containers using Iterators. Iterators are acquired using the iterator() method, and provide the hasNext() and next() methods.

1.10 Java I/O

• Output to the console is done using System.out.println(String).
1.11 **Javadoc (API) Comments**
- The Java API is documented in HTML and is available on the web.
- Useful sections are java.lang, java.io, java.util.

1.12 **The static keyword**
- Fields with the `static` keyword are called *class fields*.
  - Class fields are instantiated *once* for all objects of that class.
  - Changing a class field in 1 place changes the value for all objects of that class.
  - Class fields exist even if no object has been instantiated. They can be accessed as ClassName.field.
  - Methods with the `static` keyword are called *class methods*.
  - Class methods can be called as either `objectName.method()` or `ClassName.method()` with identical effects.
  - Class methods can only access `static` (class) fields.

1.13 **The final keyword**
- The `final` keyword is used to indicate constant constructs.
  - When used on fields, the `final` keyword indicates that the value of the field, once set, cannot be changed.
  - `final` methods cannot be overridden in derived classes. When doing a polymorphic call, the derived version of the method, if it exists, will never be used.
  - `private` methods are inherently `final`.
  - `final` classes cannot be extended (i.e. inherited from).

1.14 **abstract classes**
- `abstract` classes are halfway between normal classes and interfaces.
  - They can contain some declared, but undefined methods (like interfaces), plus defined methods and fields (like normal classes).
  - The programmer cannot instantiate `abstract` classes, so an implementation (WhateverClassImpl) class which extends the `abstract` class and defines all methods must be used.

1.15 **Resources**
- Eckel, Bruce. *Thinking in Java, 3rd Ed.* http://www.mindview.net/Books/TIJ/
- Java 1.4.1 API docs. http://java.sun.com/j2se/1.4.1/docs/api/index.html