Java Review - DPR 204

Wrapper Classes

- Each primitive type has a wrapper class that converts it to an object:
  - primitive type / Wrapper type
  - `boolean - Boolean`
  - `char - Character`
  - `byte – Byte`
  - `short - Short`
  - `int - Integer`
  - `long – Long`
  - `float – Float`
  - `double - Double`
- The wrapper classes provide fields and methods that are needed.
- One need is to be able to place primitive values in Collections. These always require objects and will not accept primitive values. Example: `coll.add(new Integer(7))`

Special Operators

- `==` tests for reference equality
  - For primitive values this is same as value equality
  - For reference types this means that both references are to the same object (same memory address)
- `!=` tests for reference inequality.
- `new` - creates a new instance of the class. Operator is followed by a constructor reference.
- `instanceof` - tests if an object is of a certain type. Example: `str instanceof String`.
- To compare string values use `equals()` method or `equalsIgnoreCase()` method.

Variables

- A variable is a name for a piece of memory. A variable must be declared and initialized before it can be used.
- Instance variable - Data member of a class that is instantiated for each object of the class.
- Static variable - Data member of a class that belongs to the class itself. Must be prefaced by the class name followed by dot.
- Local variable - variable declared in a method and is instantiated for each invocation of the method.

Example:

```
class MyClass {
  String x;      // instance variable
  static int y; // static variable

  public void f() {
    double z;    // local variable
    . . .
  }
}
```

Java Review - DPR 204

Java Source File Structure

```
// package name (optional but recommended)
package com.company.project;
// imports (only java.lang.* is automatically imported)
import java.util.*;
import java.io.*;
// class declaration
public class MyClass {
// Instance variables (zero or more)
// Constructor declarations (zero or more)
// Method declarations (zero or more)
}
```

Flow Control Statements

- Selection statements
  - Simple `if`
  - `if-else`
  - `switch` (usually uses `break` statement)
- Iteration statements
  - `for`
  - `for-each`
  - `while`
  - `do`
- Transfer statements
  - `break`
  - `continue`
  - `return`
  - `throw` (raise an exception)

ArrayList Class

- An ArrayList object is like a flexible array that grows/shrinks as per the number of elements placed within it.
- Important ArrayList methods:
  - add() - adds parameter to end of array.
  - remove(position) - removes the element at the given position and returns its value.
  - set(position, E) - replaces the element at the given position with the given element.
  - etc.

Exception Handling

- Exceptions are abnormal events that happen during program execution.
- Exceptions are objects that are derived from the `Throwable` class.

- `Throwable` has two subclasses:
  - `Error` - a subclass of Error is not recoverable
  - `Exception` - has many subclasses one of which is `RuntimeException`
- Any subclass of `Exception` except `RuntimeException` is a checked exception.
- Checked exceptions require
  - A `try/catch` block **-or-**
  - A `throws` clause in the method declaration and a `try/catch` block elsewhere
- `try / catch` are used as brackets to surround the code that can raise the exception.
- `catch(ExceptionType e)` receives the exception object if the exception is raised.
- Catch blocks can be repeated (like if/else statements) that progress from the finest to the most general exception type.
- `finally` clause, if present, is always executed whether the exception is raised or not.
- Syntax:

```
try {
   // execute statements that may raise exceptions
} catch (ExceptionType e) {
   // code here handles the exception
} finally {
   // code here usually releases resources
}
```

OO Concepts

- Object - an instance of a particular class.
- Class - a group of objects that have similar state and behavior.
- Encapsulation - a technique whereby an object's data is protected from direct access by use of getters and setters. Moreover, a technique to change a class's internal representation without affecting collaborating classes.
- Relationships:
  - Is-a - inheritance (is a subclass, is a child, is a descendent)
  - Uses - dependency (depends on, makes use of)
  - Has-a - composition (is a part of, makes up)
- Inheritance - Propagation of state and behavior to descendent classes
- Polymorphism - a technique whereby the behavior is tailored to the nature of the implementing class.

OO Keywords

- `new` - creates a new object instance.
- `this` - reference to the invoking object.
- `extends` - expresses inheritance relationship between child class and parent class.
- `implements` - adds extra behavior to class by implementing methods of an interface. Also, an inheritance relationship.

- `abstract` - expresses the fact that a concrete implementation is needed.
- `final` - expressed the prohibition on class inheritance or method overriding.
- `interface` - provides a list of abstract methods that need an implementation.
    - important interfaces: `Comparable, Iterable, Serializable.`

Constructors

- Method names that match the name of the class in which they are declared.
- Main purpose is to initialize an instance of the class to a valid state.
- Gets data for state via parameter values.
- Compiler always declares a default constructor with no parameters (but visibility can be set to private).
- Multiple constructors are possible but each must have a different signature (order and type of parameters).

Methods

- Name should be a verb, e.g. `Deck` class has a `deal()` method.
- Can return a value of any type or no value if return type is `void`.
- Multiple methods can have the same name (overloaded name) provided each method signature is different.
- Can declare formal parameters.
- Non-static methods are passed an implicit parameter called `this` which is an object of the class in which the method is declared, e.g.
  ```
  deck.deal(1);//deck object is identical to 'this' within the method
  ```
- Should not be too large (less than a screen). If larger then decompose into smaller methods.

Inheritance

- Mechanism by which new classes are derived from existing classes.
- New class inherits all data and function members from the existing class.
- Parent class declares common members; child class can specialize the parent by
    - overriding function members
    - declaring additional data and function members
- Parent, superclass, and base class are synonyms.
- Child, subclass, and derived class are synonyms.
- The `super` keyword is used to call a superclass's constructor by a subclass.

Polymorphism

- Technique to achieve different behavior from derived classes through the use of method overriding. e.g.,
  ```
  area(); // polymorphic method overridden by each shape object
  ```
- The method that is overridden is typically declared `abstract` in the parent class.

- Additional polymorphic behavior can be achieved by implementing an interface.

Object Class

- All classes implicitly extend the Object class. By doing this they inherit methods:
  - `equals()` - tests for reference equality by default, but can be overridden to test for value equality.
  - `toString`() - provides a printable string for each object.
  - etc.

Interface

- An interface is used to give a class added behavior. It can be thought of as a poor man's form of multiple inheritance. Example:

```
Card implements Comparable<Card> { ... }
```

- An interface is a list of methods. The methods can have one of these attributes:
  - `abstract` - (optional) The implementer must implement the method.
  - `static` - The method body is provided.
  - `default` - The method body is provided but can be overridden.

  If the objects of a class has a natural ordering then the class should implement the `Comparable` interface which has a single method `compareTo()`. If you do this you get sorting functionality for free.

Visibility modifiers

- To enforce encapsulation Java has four visibility modifiers (one is implicit, called "default" or "package private")

	Class	Package	Subclass (same pkg)	Subclass (diff pkg)	World
public	+	+	+	+	+
protected	+	+	+	+	
no modifier	+	+	+		
private	+				