Inheritance, Abstract Classes and Interfaces

Inheritance in Java

- Inheritance is a mechanism for defining a new class upon an existing class
  - Subclass, superclass
  - Subclass may add new states and behaviors
  - Subclass may override methods
  - Constructors are not inherited
- Inheritance allows the refinement or specialization of an existing class
  - E.g. Circle is subclass of Shape
  - Inherited members can be redefined
  - Access class of inherited members can't be narrowed
  - New members can be added
- Inheritance is the key feature of object-oriented systems to promote software reuse
- The extends keyword indicates that a class inherits state and behavior of another class
  - Java class can inherit from only one superclass (super)
  - every class implicitly extends Object class
  - final classes cannot be extended (eg. java.lang.String)
- Potentially adds new data and methods
- May redefine existing methods
- Java programming language does not permit you to extend more than one class
  - To avoid the issues of multiple inheritance of state
  - The ability to inherit fields from multiple classes
  - The ability to inherit method definitions from multiple classes

Method Inheritance

- A subclass inherits members from the superclass
  - All non-private methods and variables of the parent are inherited
- Locally defined method hides method in parent
  - both methods are nevertheless available for the class
- Use the keyword super to explicitly refer to the superclass method

Constructor of Superclasses

- Subclasses must construct the superclass
    - Automatic if the superclass has a default constructor
    - Otherwise, must use super() to call a specific constructor
    - super() must be the first statement in the constructor

Casting

- A variable can refer to an object of its own type or one of a subtype of its own type
- Upcasting: object returned by the new operator is implicitly casted to a superclass object before the assignment
- Downcasting: explicitly casting a reference of a superclass to one of its subclass
    - if the first assignment doesn't refer to a subclass: ClassCastException, use instanceof operator to avoid

Final Modifier

- The final modifier on classes
    - Prevents the class from being extended
- The final modifier on methods
    - Prevents a method being overridden in derived classes
- The final modifier on variables
    - Makes the "variable" constant
    - Fixes the value of a variable for its lifetime
    - Must be supplied with an initial value

Method Overriding

- Overriding occurs when the subclass defines the same method as the superclass did
    - Method signatures must be identical
    - Returned value type must be identical
    - Access class cannot be narrowed
- Virtual Machine uses dynamic lookup to determine which method to invoke
    - The method called depends on the datatype of the target object

- Create objects of different (but related) types and manage them all in the same way
    - The code that manages the objects and sends messages to them does not need to know the exact datatypes of the individual objects

Abstract Classes

- An abstract class is a class that is declared abstract
    - It may or may not include abstract methods
    - Cannot be instantiated
    - Can be subclassed.
- An abstract method is a method that is declared without an implementation (without curly brackets and followed by a semicolon)
- If a class includes abstract methods, then the class itself must be declared abstract
- When an abstract class is subclassed:
    - The subclass usually provides implementations for all of the abstract methods in its parent class
    - If it does not, then the subclass must also be declared abstract
- Consider using abstract classes if:
    - You want to share code among several closely related classes
    - You expect that classes that extend your abstract class have many common methods or fields
    - You want to declare non-static or non-final fields, this enables you to define methods that can access and modify the state of the object to which they belong
- Abstract class may have static fields and static methods

Interfaces

- Interfaces: abstract types with no implementation
    - All methods are implicitly public
    - All methods are implicitly abstract - no default behavior
    - All variables are implicitly public static final - must be initialized
- Unlike abstract classes, interfaces reside in a separate inheritance hierarchy
    - Classes can implement multiple interfaces
    - Interfaces can be extended (interface inheritance)
- Consider an interface that you have developed
    - You want to add an additional method to it
    - All classes that implement the old interface will break because they no longer implement the old interface

- Extending interface
- Multiple inheritance is supported

Interface vs. Abstract Classes

- Use abstract class if
    - You will plan on using inheritance since it provides a common base class implementation to derived classes
    - You want to be able to declare non-public members (in an interface, all methods must be public)
    - You think you will need to add methods in the future
- Use interface if
    - You think that the API will not change for a while (so like never)
    - You want to have something similar to multiple inheritance