CMPE212 – Reminders

• Assignment 3 due Friday.

• Quiz 2 next week in the lab. Topic Coverage in yesterday’s lecture.

Today

• Abstract Classes.

• Interfaces.

abstract Classes

• It is not unusual to declare a class in the root of a hierarchy to be
  abstract:

  public abstract class MyClass …

• Any class declared this way cannot be instantiated.
• It can only be extended.
• Unlike an interface, an abstract class can also contain
  concrete method definitions and any kind of attribute.
• If a class has one or more abstract methods, the class must be
  declared abstract as well.

Aside – Preventing Instantiation, Cont.

• As mentioned earlier: write a private default constructor.
• But, you can also name any class as abstract, even if it does not
  have any abstract methods. This will also make sure that this class
  cannot be instantiated.
• The advantage here is that you can now have public constructors
  that can be used by child classes. A non-abstract child class can be
  instantiated.

abstract Classes, Cont.

• abstract methods have no code in them. For example:

  public abstract String getListing();

• A class that extends an abstract class must override all the
  abstract methods in the class, unless it wants to be abstract too.
• Overriding means that you must have a concrete implementation of
  that method in the child class.
• Unlike an interface, you need to write public abstract for each
  abstract method signature.

abstract Classes, Cont.

• Why bother?

• An abstract class forces sub-classes to define certain methods. This
  helps ensure that the hierarchy has a consistent design.
• Also, when declaring a method in a very abstract class, then you
  don’t have to worry about what to do in the method body, especially
  if it must return a value.
• One way to provide the mechanism for Polymorphism!
Aside – Polymorphism (A Quick Peek)

- We will learn more about this important aspect of OOP later, but for now:
- Polymorphism is when a pointer of a parent class type ends up pointing to different child class objects at runtime. Also called Dynamic Binding. The process must also satisfy early binding:
  - Early Binding is satisfied when the parent class also owns the method that will end up being invoked from the morphed child class objects. The use of interfaces and abstract classes can make for easier coding to satisfy early binding.

Interfaces in Java 7 and Older

- Interfaces used to just contain:
  - constant attributes and/or
  - abstract methods.
- Abstract methods consist of just the method header – there is a semi-colon instead of {}.
  - (Abstract classes can contain both concrete and abstract methods, along with any kind of attribute.)

Interfaces in Java 8

- In addition to the above, we can also have:
  - Default methods.
  - Static methods.

Java 8 Interface Default Methods

- Must have the keyword default at the beginning of the method header.
- These methods will not be empty, but can contain implementation code.
- This will be a “default” implementation for this method.
- This method can be overridden or inherited by a class implementing the interface, or by another interface extending this interface.
  - (Yes, you can have an interface hierarchy!)

Java 8 Interface static Methods

- They behave just the same way a static method behaves in a normal class.
- A static method is not inherited.
- Also simplifies design.
Interfaces in Java 9

- In addition to the options listed above, Java 9 interfaces can have private static and normal private methods, just like those in a normal class.
- The purpose is to simplify the construction of the interface by providing a means to eliminate common code in other methods.

Interfaces in Java 9, Cont.

- To summarize, an interface can now contain:
  - public static final attributes.
  - abstract methods.
  - default methods.
  - static methods.
  - private methods.
  - private static methods.

- The keywords public and abstract are still not required.
- What is not allowed?

Interfaces, Cont.

- Interfaces do not extend Object.
- Interfaces can extend multiple interfaces (but not other classes).
- Classes can implement one or many interfaces:

```java
class Test implements interface1, interface2, interface3, ...
```  
- If two interfaces have the same method name, the implementing class must implement that method, even if it is a default method.

Interfaces, Cont.

- Attributes in interfaces can only be public final static, you cannot use private. You don’t even have to specify public final static as it is assumed. Constants must be initialized.

- In Java 8 and 9, you should not and sometimes cannot use the public and abstract keywords in method headers. They are not needed anyways...
Interfaces, Cont.

- A class that implements an interface must have a concrete implementation of every abstract method signature in the interface.
- The class can just accept (inherit) default methods, can use static methods and will not see private methods.
- When designing an interface, you must try to use unique method and constant names, in case your interface becomes part of a multiple implementation.

The Comparable Interface in Java

- Without all the javadoc comments:

```java
package java.lang;
public interface Comparable<T> {
    public int compareTo(T o);
}
```
- That's it!! This is a Generic Interface now.

Comparable Interface, Cont.

- A class that implements Comparable can be sorted with:
  - `Arrays.sort()`, or
  - `Collections.sort()`, or
  - `from ArrayList.sort()`

- The first two methods are already contained in Java (package `java.util`) and use very fast Mergesort or Quicksort algorithms.
- Why is this implementation necessary?

Implementation in Arrays.mergesort()

- For example, this is a line of code from the mergeSort() method in the java.util.Arrays class:

  ```java
  if (((Comparable)src[mid-1]).compareTo(src[mid]) <= 0)
  ```

- This method sorts `src`, which is of type: `Object[]`.
- So, provided your Object implements Comparable, this method can sort it!

Interfaces, Cont.

- While interfaces are not proper objects in Java, you can "pretend" that they are in some circumstances.
- For example, a class can use the interface as an object type, and write code as if you are invoking a method declared in the interface. You can pretend that the specification is an object.
- Part of polymorphism.
- An instance of a class that implements an interface can be cast to that interface type.

Another Example on Interfaces

- Look at DoRunRunDemo and see how it works.
- The example contains examples of how the interface can (and cannot) be used in Java 8 and 9.
Summary of Interface Demo

• Inside the interface:
  – Can only have public final static attributes.
  – Can have public abstract, public static, public default, private and
    private static methods.
  – Do not use the public or abstract access modifiers.
  – Public static methods can invoke private static methods.
  – Public default methods can invoke private non-static methods.

Summary of Interface Demo, Cont.

• Inside a class implementing the interface:
  – Must implement all abstract methods (or this class will be abstract).
  – Constants are inherited.
  – Default methods can be inherited, made abstract (in which case
    the class will be abstract) or overridden.
  – Public static methods can only be invoked from the interface
directly. They are not inherited.

Summary of Interface Demo, Cont.

• Inside some other class:
  – Constants and public static methods can be accessed directly
    through the interface.
  – The interface cannot be instantiated.
  – A non-abstract class implementing the interface can be
    instantiated.
  – Pointers can be declared to be of the interface type and they can
    then point to instances of a class that implements the interface.
  – A class implementing the interface can be cast to be of the
    interface type.