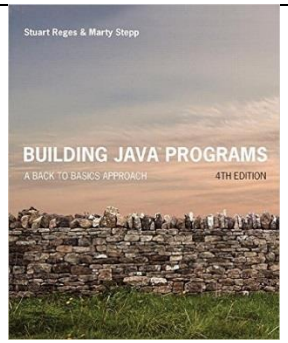
Abstract Classes and Interface

Reading Assignment: Read Chapters 9.
Inheritance and Interface –
Building Java Programs (Stuart Reges and Marty Stepp)



Abstract Classes

- The abstract class is declared using the keyword abstract. An abstract class is a placeholder in a class hierarchy that represents a generic concept.

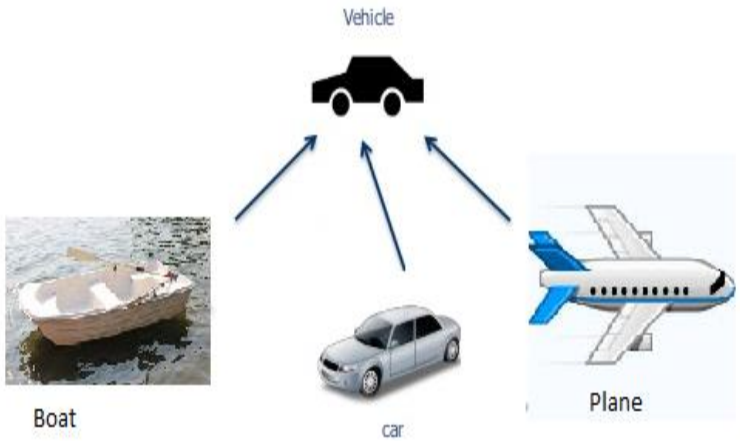
- An abstract class, in the context of Java, is a superclass that cannot be instantiated and is used to state or define general characteristics. An object cannot be formed from a Java abstract class.

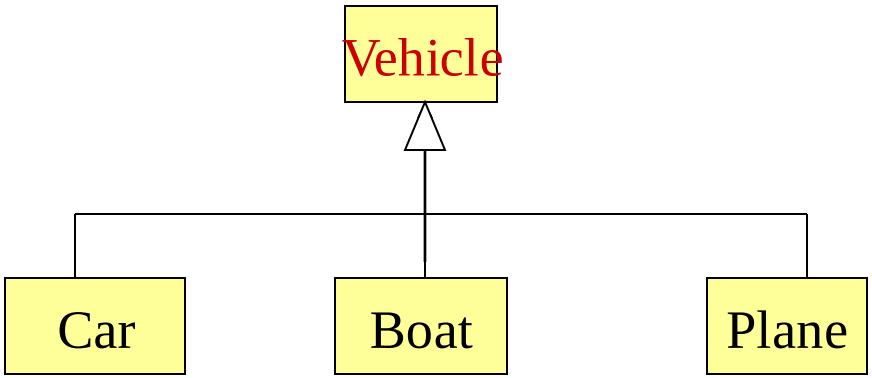
Abstract Classes

- Java allows abstract classes
  - use the modifier `abstract` on a class header to declare an abstract class

```
abstract class Vehicle
{ … }
```

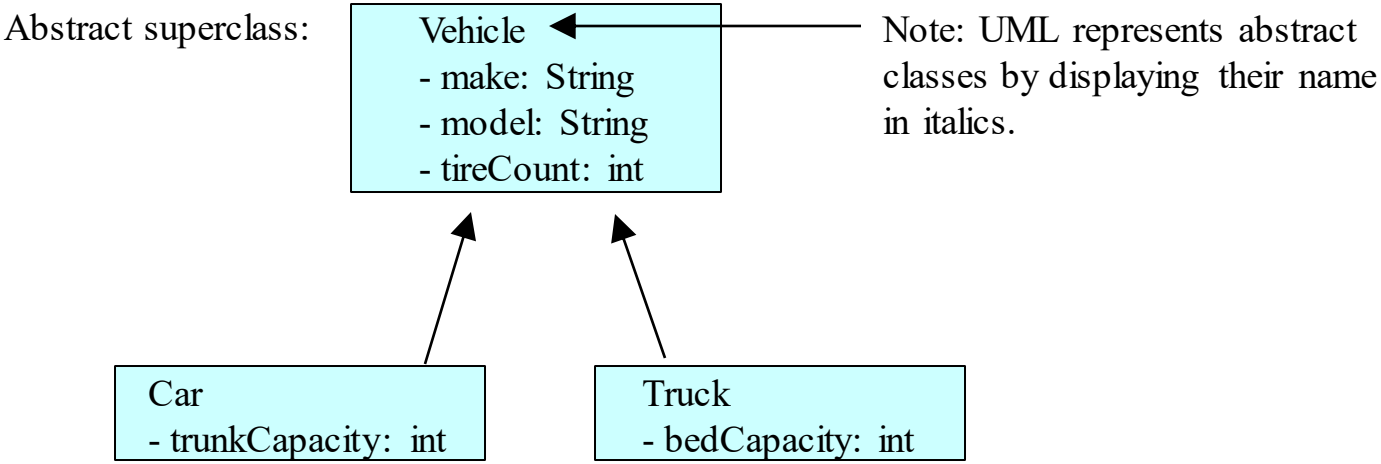
- An abstract class is a placeholder in a class hierarchy that represents a generic concept





Abstract Class Example

- In the following example, the subclasses represent objects taken from the problem domain.
- The superclass represents an abstract concept that does not exist "as is" in the real world.



Abstract Class: Example

- An abstract class often contains *abstract methods*, though it doesn't have to
  - Abstract methods consist of only methods *declarations*, without any method body

```
public abstract class Vehicle
{
   String name;

   public String getName()
          { return name; } \\ method body

   abstract public void move();  \\ no body!
}
```

Abstract Classes

- An abstract class often contains *abstract methods*, though it doesn't have to
  - Abstract methods consist of only methods *declarations*, without any method body

- The non-abstract child of an abstract class must override the abstract methods of the parent
- An abstract class cannot be instantiated (why?)

- The use of abstract classes is a design decision; it helps us establish common elements in a class that is <u>too general to instantiate</u>

Interfaces

Java Interface

- A Java *interface* is a collection of constants and abstract methods
  - abstract method: a method header without a method body; we declare an abstract method using the modifier `abstract`
  - since all methods in an interface are abstract, the `abstract` modifier is usually left off

- Methods in an interface have public visibility by default

Interface: Syntax

interface is a reserved word

```
public interface Doable
{
   public static final String NAME;

   public void doThis();
   public int doThat();
   public void doThis2 (float value, char ch);
   public boolean doTheOther (int num);
}
```

A semicolon immediately follows each method header

No method in an interface has a definition (body)

Implementing an Interface

- A class formally implements an interface by
  - stating so in the class header in the `implements` clause
  - a class can implement multiple interfaces: the interfaces are listed in the implements clause, separated by commas

- If a class asserts that it implements an interface, it must define all methods in the interface or the compiler will produce errors

Implementing Interfaces

```
public class Something implements Doable
{
    public void doThis ()
    {
        // whatever
    }

    public void doThat ()
    {
        // whatever
    }

    // etc.
}
```

implements is a reserved word

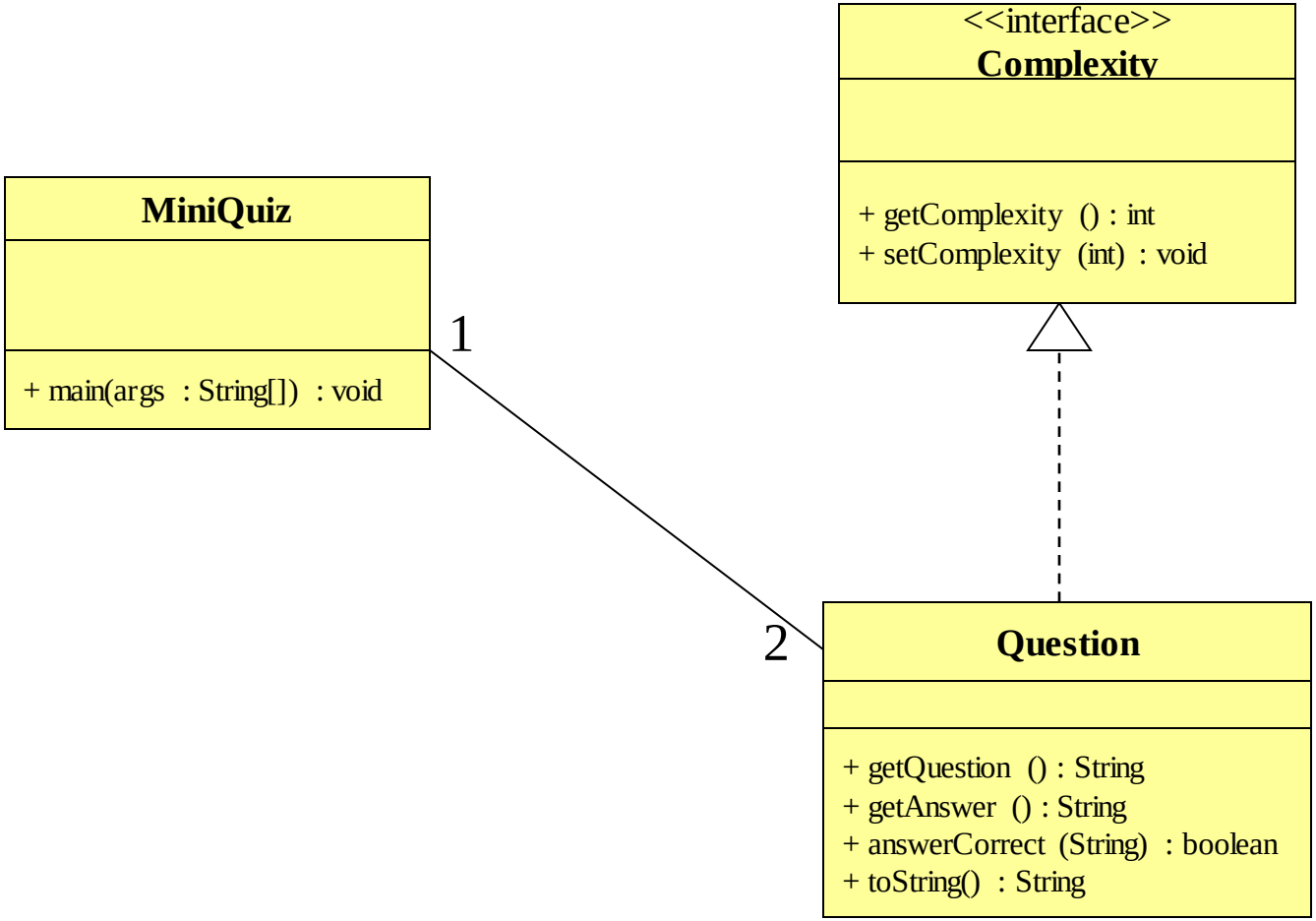
Each method listed in Doable is given a definition

```
public class ManyThings implements Doable, AnotherDoable
```

Interfaces: An Example

- A class that implements an interface can implement other methods as well

UML Diagram


<<interface>>
Complexity
+ getComplexity () : int
+ setComplexity (int) : void
MiniQuiz
+ main(args : String[]) : void
1
2
Question
+ getQuestion () : String
+ getAnswer () : String
+ answerCorrect (String) : boolean
+ toString() : String

Interfaces: Examples from Java Standard Class Library

- The Java Standard Class library defines many interfaces:
  - the `Iterator` interface contains methods that allow the user to move through a collection of objects easily
    - `hasNext(), next(), remove()`
  - the `Comparable` interface contains an abstract method called `compareTo`, which is used to compare two objects

```
if (obj1.compareTo(obj2) < 0)
    System.out.println("obj1 is less than obj2");
```

Polymorphism via Interfaces

- Define a polymorphism reference through interface
  - declare a reference variable of an interface type

```
Doable obj;
```

  - the `obj` reference can be used to point to any object of any class that implements the `Doable` interface
  - the version of `doThis` depends on the type of object that `obj` is referring to:

```
obj.doThis();
```

More Examples

```
Speaker guest;

guest = new Philosopher();
guest.speak();

guest = Dog();
guest.speak();
```

```
Speaker special;
special = new Philosopher();

special.pontificate(); // compiler error
```

```
Speaker special;
special = new Philosopher();

((Philosopher)special).pontificate();
```

```
public interface Speaker
{
   public void speak();
}

class Philosopher extends Human
   implements Speaker
{
   //
   public void speak()
   {…}
   public void pontificate()
   {…}
}

class Dog extends Animal
   implements Speaker
{
   //
   public void speak()
   {
   …
   }
}
```

Using an Interface as a Type

- "When you define a new interface, you are defining a new reference data type.
  - "You can use interface names anywhere you can use any other data type name.
  - "If you define a reference variable whose type is an interface, any object you assign to it *must* be an instance of a class that implements the interface." [http://docs.oracle.com/javase/tutorial/java/IandI/interfaceAsType.html]
- Example on the next slide:
  - A method for finding the largest object in a pair of objects, for *any* objects that are instantiated from a class that implements **`Relatable`**.

```
public interface Relatable {
   public int isLargerThan( Relatable other );
}
```

Using an Interface as a Type

```
public Object findMax(Object object1, Object object2) {
  Relatable obj1 = (Relatable)object1;
  Relatable obj2 = (Relatable)object2;
  if( (obj1).isLargerThan(obj2) > 0 )
    return object1;
  else
    return object2;
}
```

- If comparisons are important in your application, then you'll be able to write very elegant code!
  - You can write `z.findMax(x, y)`, if `x` and `y` are instances of any class which extends `Relatable`.

Interface Hierarchies

- Inheritance can be applied to interfaces as well as classes
- One interface can be used as the parent of another
- The child interface inherits all abstract methods of the parent
- A class implementing the child interface must define all methods from both the parent and child interfaces
- Note that class hierarchies and interface hierarchies are distinct (they do not overlap)

Abstract Classes vs Interfaces

- When should one use an Abstract class instead of an interface?
  - If the subclass-superclass relationship is genuinely an "is a" relationship.
  - If the abstract class can provide an implementation at the appropriate level of abstraction.
  - If use an abstract class to provide default behaviour.

- When should one use an interface in place of an Abstract Class?
  - When an absolute contract is really intended.
  - When the methods defined represent a small portion of a class.
  - When you cannot reasonably implement any of the methods.
  - When use for multiple inheritance.

Vocabulary

- abstract method—a method which is declared but not defined (it has no method body)
- abstract class—a class which either (1) contains abstract methods, or (2) has been declared abstract
- interface—similar to a class, but contains only abstract methods (and possibly constants)
- **adapter class**—a class that implements an interface but has only empty method bodies
  (example: **java.awt.event**, **java.awt.dnd** and **javax.swing.event** package)

Interfaces in Java 8

(Backup Slide)

- In Java 8, an interface may contain
  - `default` implementations of instance methods, and
  - implementations of `static` methods.
- In any OO language, an interface
  - cannot be instantiated, and
  - defines a "contract" which any realization of the interface must fulfil.
- Java is a strongly-typed language.
  - Java compilers *can* enforce contracts, by refusing to compile classes whose implementations might "partially realize" an interface.
- Java is a tightly-specified language.
  - If a compiler allows instantiations of incompletely-implemented interfaces, then it is *not* a Java compiler.