CS 213 – Software Methodology
Spring 2016

Lecture 6 – Feb 4

Interfaces – Part 1

Comparing for inequality in an algorithm implementation

```
public class Searcher {
   ...
   public static<T> boolean
   binarySearch(T[] list, T target) {
      ...
      list[index].___?___target

      ...
   }
   ...
}
```

How to compare for inequality? All we know Is `T` is some `Object`, but `Object` does not define an inequality comparison method

Need to have a type definition for parameters that will guarantee the existence of a method that can be used for inequality comparison

Comparing for inequality in an algorithm implementation

```
public class Searcher {
   ...
   public static<T> boolean
   binarySearch(T[] list,
                    T target) {
       ...
       list[index].___?___target
       ...
   }
}
```

Solution is to use a pre-existing interface that is known to prescribe an inequality comparison method.

Or, define an appropriate interface if none exists.

The interface *introduces a type* that can be checked by the compiler for match between caller and callee

e.g. `java.lang.Comparable` interface, which defines a `compareTo` method

```
public static
<T extends Comparable<T>>

    list[index].compareTo(target)
```

Type `T` is not just any class, but one that implements the `java.lang.Comparable` interface, or extends a class (any number of levels down the inheritance chain) that implements this interface

Interfaces

The term "interface" generally refers to the means by which an object can be manipulated by its clients – in this sense the public methods of an object comprise its implicit interface.

For example, public methods `push`, `pop`, `isEmpty` (as well as constructors) in a `Stack` implicitly define its interface – these methods/constructors will be used by clients to create and manipulate stacks

Java provides a way (keyword `interface`) to define an explicit interface that can be implemented (keyword `implements`) by classes

```
public interface I { . . . }
public class X implements I { . . . }
```

Interfaces

The properties of interfaces:

- An interface is generally public
- An interface defines a new type name that is tracked by the compiler
- All fields in an interface are implicitly public, static, and final (constants)
- All methods (prescribed) in an interface are implicitly public and abstract (no method body), unless it's a default method (as of Java 8)
- When a class implements an interface, it must implement every single method that is prescribed in the interface, as public but NOT abstract
- An interface may be generic
- As of Java 8, an interface may have default methods (with implementation!), as well as static methods

Interface defined in `java.lang` package

```
public interface Comparable<T> {
   int compareTo(T o);
}
```

Prescribes a single, `compareTo` method, but there is no method body, just a semicolon terminator

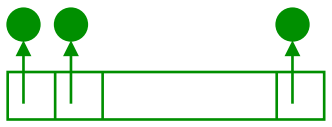
Keywords `public` and `abstract` are omitted by convention (redundant if written)

Using `java.lang.Comparable`

```
public class Point
  implements Comparable<Point> {
     . . .

public int compareTo(Point other) {
     int c = x - other.x;
     if (c == 0) {
         c = y - other.y;
     }
     return c;
 }

}
```

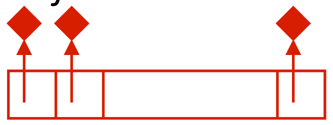
```
public class Widget
  implements Comparable<Widget> {
   . . .

public int compareTo(Widget other) {
     float f = mass - other.mass;
     if (f == 0) return 0;
     return f < 0 ? -1 : 1;
  }

}
```

Array of `Point` *objects*



target `Point`

Array of `Widget` *objects*



target `Widget`

```
public static <T extends Comparable<T>>
   boolean binarySearch(T[ ] list,
                        T target) {
          ...
          int c = target.compareTo(list[i]);
          ...
   }
}
```

Interface `javafx.event.EventHandler`

```
public interface EventHandler<T extends Event> {
   void handle(T event);
}
```

`javax.scene.control.ButtonBase` defines this method:

```
public void setOnAction(EventHandler<ActionEvent> value) {
   ...
}
```

The parameter to this method is any object that implements the `EventHandler<ActionEvent>` interface.

`javax.scene.control.Button` is a subclass of `ButtonBase`:

```
f2c.setOnAction(new EventHandler<ActionEvent>() {
    public void handle(ActionEvent e) {...}
});
```

Anonymous class that implements the `EventHandler<ActionEvent>` interface

Object created by calling the default constructor of the anonymous class

Key Points

- An interface introduces a new type (just like a class does)
- By having a class implement an interface, a specific role can be attributed to it – the role is defined by the methods prescribed by the interface (e.g. inequality comparison)

Using Interfaces:
To Define a Specialized Role For Classes

Often,
  a specialized role needs to be specified
    for some classes in an application (e.g. comparing for ==, >, <),
  and given a type name (.e.g. `Comparable`)

The type name is the interface name,
  and the role is the set of interface methods.

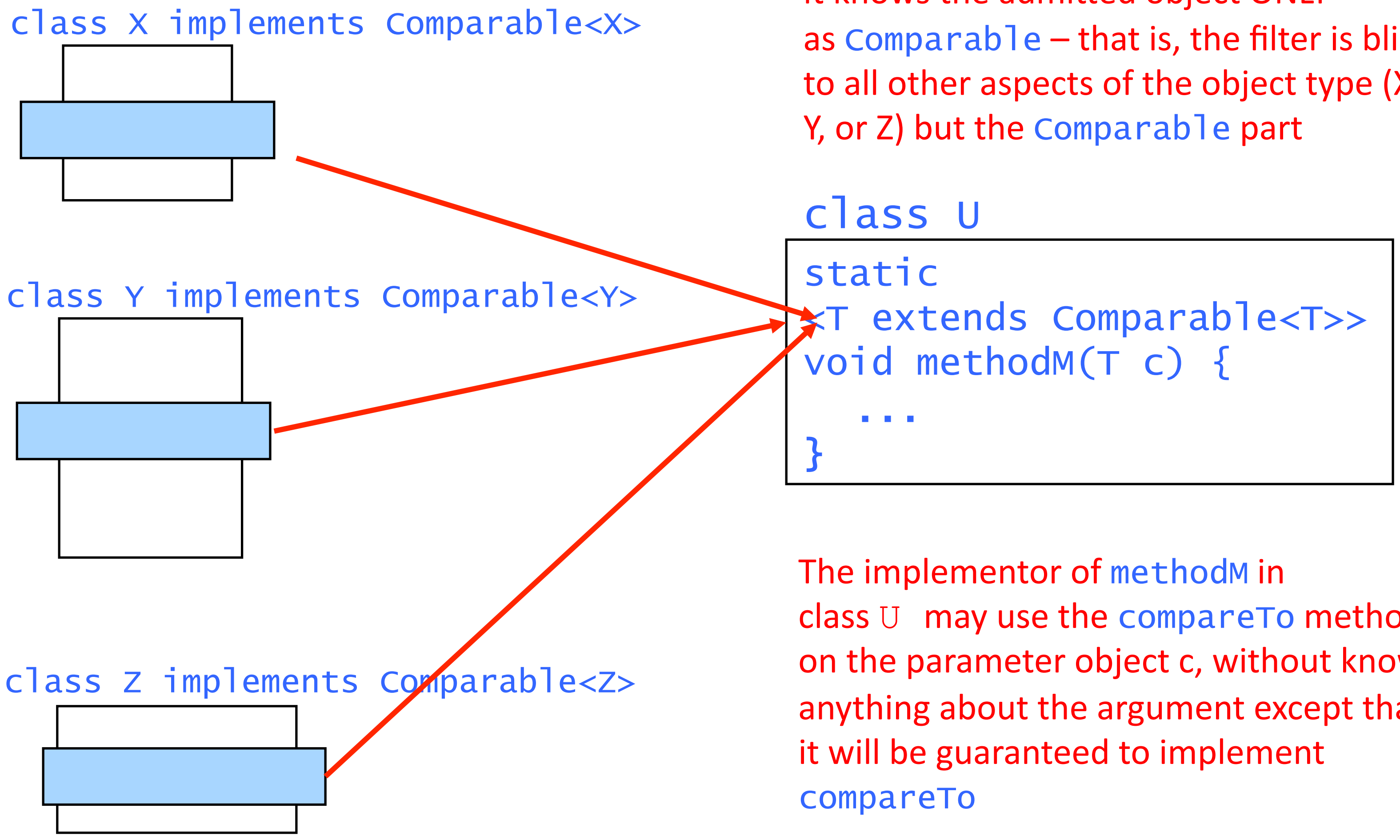
You can think of an interface as
  a filter that is overlaid on a class.

Depending on the context,
  the class can be fully itself (class type)
  or can adopt a subset, specialized role (interface type)

Specialized Role For Classes

```
public interface Comparable<T> {
    int compareTo(T o);
}
```

methodM will admit any object, so long as it is Comparable, and it knows the admitted object ONLY as Comparable – that is, the filter is blind to all other aspects of the object type (X, or Y, or Z) but the Comparable part


class X implements Comparable<X>
class Y implements Comparable<Y>
class Z implements Comparable<Z>

```
class U
static
<T extends Comparable<T>>
void methodM(T c) {
   ...
}
```

The implementor of methodM in class U may use the compareTo method on the parameter object c, without knowing anything about the argument except that it will be guaranteed to implement compareTo

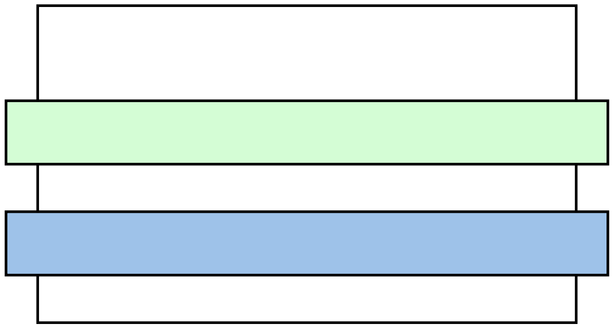
Interface to Define Specialized Role for Classes: Example 2

ebooks provide very different functionality than videos.
However you can **play** both (go through a book page by page) on a computer, and **store** both on disk.

Playing and storing are two specialized roles that are shared by EBook and Video:

```
public interface Playable { … }

public interface Storable { … }
```

```
class EBook implements
      Playable, Storable
```



```
class Video implements
      Playable, Storable
```