# CS 213 – Software Methodology

## Spring 2017

*Sesh Venugopal*

Lecture 8 – Feb 9

Interfaces – Part 2

# Interfaces

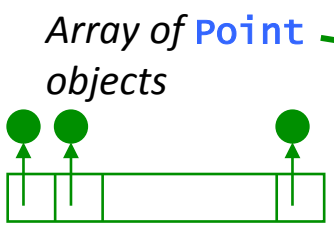
Properties of interfaces:

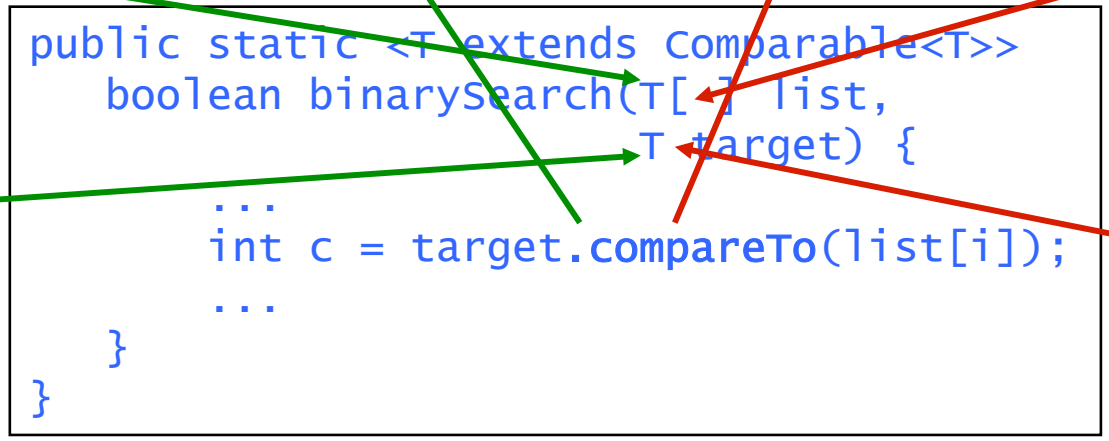
1. An interface defines a new type that is tracked by the compiler
2. All fields in an interface are implicitly public, static, and final
3. Prior to Java 8, all interface methods were implicitly public and abstract (no method body)
4. As of Java 8, interfaces can also include default and static methods (with method body) – these need to be public
5. When a class implements an interface, it must implement every single abstract method of the interface
6. An interface may be generic

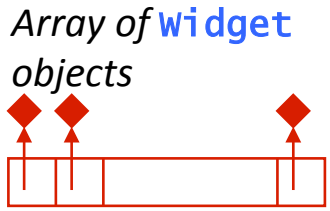
# Using java.lang.Comparable

```
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;
    }
}
```





# 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.

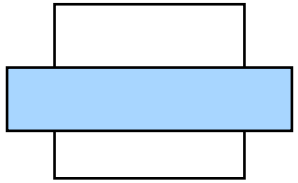
You can think of an interface as
a filter that is overlaid on a class.

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

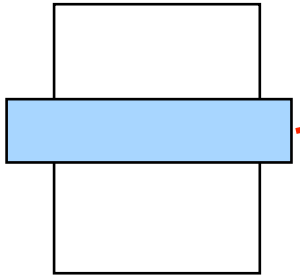
# Specialized Role For Classes

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

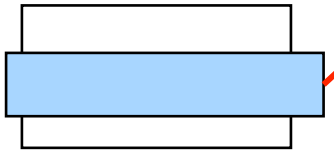
class X implements Comparable<X>



class Y implements Comparable<Y>



class Z implements Comparable<Z>



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 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

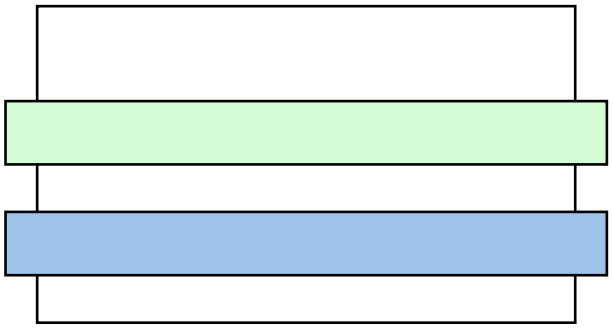
ebooks provide very different functionality than videos.
However you can **play** both (go through a book page by page) on a computer, and **store** both on disk.

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
```