 

**Advanced Placement Computer Science**

**Unit 3: Object Oriented Programming**

Lesson: Interfaces

*Last Updated: 11/12/2015*

USB is really cool, isn’t it?
 

It allows us to plug a thumb drive into any computer that supports USB, and voila, after installing the device driver, you’ve got a removable drive!!!!

Java has a similar concept, **interfaces** are a collection **of methods that must be implemented by a class, a promise from programmer to programmer!**

Let’s look at a useful interface.

public interface MouseListener extends EventListener {

 /\*\*

 \* Invoked when the mouse button has been clicked (pressed

 \* and released) on a component.

 \*/

 public void mouseClicked(MouseEvent e);

 /\*\*

 \* Invoked when a mouse button has been pressed on a component.

 \*/

 public void mousePressed(MouseEvent e);

 /\*\*

 \* Invoked when a mouse button has been released on a component.

 \*/

 public void mouseReleased(MouseEvent e);

 /\*\*

 \* Invoked when the mouse enters a component.

 \*/

 public void mouseEntered(MouseEvent e);

 /\*\*

 \* Invoked when the mouse exits a component.\*/

 public void mouseExited(MouseEvent e);

}

All methods in an interface must be **implemented by the class that implements interface**

Differences between interfaces and classes

1. **There is NO logic or commands in an interface, only method signatures**
2. **Everything is public in an interface, nothing private**
3. **NO variables in an interface**

Interfaces guarantee that certain methods are implemented. This helps write more general purpose code.

Some popular interfaces;

|  |  |
| --- | --- |
| **Interface** | **Methods that must be implemented** |
| ActionListener | public void actionPerformed(ActionEvent blah) |
| Comparable | public int compareTo(Object o) |
| KeyListener | public void keyPressed (KeyEvent e) public void keyTyped (KeyEvent e) public void keyReleased (KeyEvent e) |
| MouseListener | public void mousePressed(MouseEvent event) public void mouseClicked(MouseEvent event) public void mouseReleased(MouseEvent event)public void mouseEntered(MouseEvent event) public void mouseExited(MouseEvent event)  |

Now if we create a class that implements BOTH MouseListener and KeyListener, what do we know?

public class MyFrame implements KeyListener, MouseListener {

 //**Must implement 3 methods from KL and 5 from MouseListener**

}

How about the Comparable interface?

You must define how to **compare two objects of your class and rank them**

Works like the mathematical operator **subtraction**

For example, String implements Comparable, so

String s1 = “CAT”, s2 = “DAD”;

int result = s1.compareTo(s2); //what is the value of result??? **Since C= 67 in ASCII, and D is 68, this method returns 67-68 which is -1**


How about “Carly”.compareTo(“Allen”); **67-65 which is 2**

How about “Eric”.compareTo(“Erin**”); E’s are same so we go to r versus r same so i versus i same so c versus n** **99-114 which is -15**

How about “Sanjee”.compareT0(“Sanjee”); **0**

How about “tea”.compareTo(“teapot”); **0-112 = -112**

Summary: **0-112 = -112**

What does this allow us to do with Strings?

**This allows us to compare strings alphabetically in English and sort them(NOTE: all UPPERCASE comes before all lowercase and space is ascii 32 so it is below them all)**

Furthermore, there is a method **Collections.sort** which will sort any ArrayList of Comparables

ArrayList<String> namesAL = new ArrayList<String>();

 namesAL.add("Cara");

 namesAL.add("Carly");

 namesAL.add("Bob");

 namesAL.add("Austin");

 namesAL.add("Zoe");

 namesAL.add("Jessica");

 Collections.sort(namesAL);

 for (int i = 0; i < namesAL.size(); i++) {

 System.out.println(namesAL.get(i));

 }

//OUTPUT
Austin

Bob

Cara

Carly

Jessica

Zoe

How about a TennisPlayer class that implements Comparable?

import java.util.Date;

public class TennisPlayer implements Comparable {

 private String lastName, firstName, countryOrigin;

 private Date dob;

 private int rank, annualEarnings, tourPoints;

 public int getRank() { return rank; }

 public int compareTo(Object other) {

 TennisPlayer oth = (TennisPlayer) other; //must TYPECAST when using compareTo

 return rank – oth.getRank();

 }

}

**How do we use this to test 2 players?**

public class Tester {

 public static void main(String args[]) {

 TennisPlayer rafa = new TennisPlayer(“Nadal”, “Rafael”, “Espana”, new
 Date(1986,6,3), 5, 583074, 6675);

 TennisPlayer roger = new TennisPlayer(“Federer”, “Roger”, “Switzerland”, new
 Date(1981,8,18), 3, 550,017, 5205);

 //Write the logic to compare the two players

 **if(rafa.compareTo(roger)==0){**

 **System.out.println(“Equal”);**

 **}
 else if(rafa.compareTo(roger)<0){
 System.out.println(“Rafa ranked lower number”);
 }
 else {
 System.out.println(“Rafa ranked higher number”);
 }**

 }

}

**Interfaces can also be passed as parameters to a method.**

/\*\*

 \* The <code>AudioClip</code> interface is a simple abstraction for playing a sound clip. Multiple

 \* <code>AudioClip</code> items can be playing at the same time, and the resulting sound is mixed together to

 \* produce a composite.

 \* @author Arthur van Hoff

 \* @since JDK1.0

 \*/

public interface AudioClip {

 /\*\*

 \* Starts playing this audio clip. Each time this method is called,

 \* the clip is restarted from the beginning. \*/

 void play();

 /\*\*

 \* Starts playing this audio clip in a loop. \*/

 void loop();

 /\*\*

 \* Stops playing this audio clip.\*/

 void stop();

}

Write a method called playIntro that takes in an AudioClip (ac)reference and plays it as the intro to a program
**public void playIntro(AudioClip snd) {
 snd.play();
}**