



TECHNISCHE
UNIVERSITÄT
WIEN

Recap - Inheritance

➤ java.lang.Object is the top class

```
class Machine {  
    // implicit superclass if not specified otherwise  
}
```

```
public class Main {  
    public static void main(String[] args){  
        Machine machine1 = new Machine();  
        Machine machine2 = new Machine();  
        // can we do anything with the machine(s) here?  
    }  
}
```

Recap - Inheritance

➤ `java.lang.Object` is the top class

- Certain functionality is already provided

`machine1.`

```
m equals(Object obj) boolean
m hashCode() int
m toString() String
m getClass() Class<? extends Machine>
m notify() void
m notifyAll() void
m wait() void
m wait(long timeout) void
m wait(long timeout, int nanos) void
cast ((SomeType) expr)
field myField = expr
inst expr instanceof Type ? ((Type) expr). : null
instanceof expr instanceof Type ? ((Type) expr). : null
lambda () -> expr
```

Recap - Inheritance

- Abstract from details: e.g. a caller does not need to know about the internals

Example: print any object using toString().

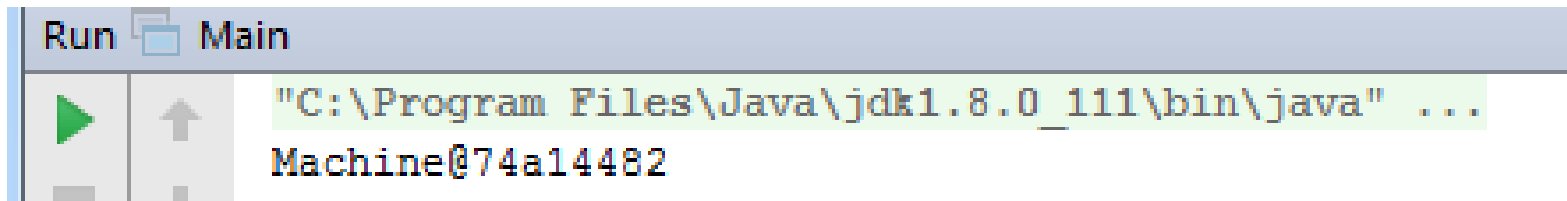
toString method is often very usefull to debug your applications

```
System.out.println(machine1.toString());
```

Output ? /* recal that we did not defined any toString method in our Machine class */

Recap - Inheritance

```
System.out.println(machine1.toString());
```



The screenshot shows a Java IDE console window. The title bar reads "Run Main". The console output is as follows:

```
"C:\Program Files\Java\jdk1.8.0_111\bin\java" ...  
Machine@74a14482
```

In the above example, printing machine1 prints the class name and the unsigned hexadecimal representation of the hash code of the object

Remember, overriding the toString() method, returns the desired output. You can create a custom string representation of your object. This enables you to identify your object(s) (helpful for debugging).

Recap – Inheritance (Class Integer)

Example: Class Integer and toString method(s)

java.lang.Object

java.lang.Number

java.lang.Integer

All Implemented Interfaces:

Serializable, Comparable<Integer>

Let's inspect toString method which **overrides** toString method in class Object

Recap - toString (Class Integer)

Example: Class Integer and toString method(s)

Modifier and Type
String

Method and Description
toString()

Returns a String object representing this Integer's value.

static String

toString(int i)

Returns a String object representing the specified integer.

static String

toString(int i, int radix)

Returns a string representation of the first argument in the radix specified by the second argument.

[https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html#toString\(\)](https://docs.oracle.com/javase/7/docs/api/java/lang/Integer.html#toString())

Recap - toString (Class Integer)

Example: toString() from Integer class

Integer x = 5;

Note, x stores reference to an Integer object. We can use object's method(s) now.

```
System.out.println(x.toString() + '=' + "five");
```

Output ?

Recap - toString (Class Integer)

Example: toString() from Integer class

Integer x = 5;

Note, x stores reference to an Integer object. We can use object's method(s) now.

```
System.out.println(x.toString() + '=' + "five");
```

Output 5=five

Recap - Java pitfalls for beginners

```
int x1 = 2;
```

```
int x2 = 5;
```

```
int sum = x1 + x2;
```

```
System.out.println(x1 + '+' + x2 + '=' + sum);
```

Output ?

Recap - Java pitfalls for beginners

```
int x1 = 2;  
int x2 = 5;  
int sum = x1 + x2;  
System.out.println(x1 + '+' + x2 + '=' + sum);
```

Output 118 // Why?

Recap - Java pitfalls for beginners

```
int x1 = 2;  
int x2 = 5;  
int sum = x1 + x2;  
System.out.println(x1 + '+' + x2 + '=' + sum);
```

Output 118 // Why?

Special support for the string concatenation operator
(+) <http://docs.oracle.com/javase/6/docs/api/java/lang/String.html>

Use single quotes for literal char s, double quotes for literal String s, like so: char c = 'a'; String s = "hello"; They cannot be used any other way around (like in Python, for example)

Recap - Java pitfalls for beginners

```
int x1 = 2;  
int x2 = 5;  
int sum = x1 + x2;  
System.out.println(x1 + '+' + x2 + '=' + sum);
```

Output 118 // Why?

The int value of char + (which is 43) was added to x1 instead of being concatenated, and the same holds for char = (with value 61). See ASCII Table

<https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html>

Recap - Java pitfalls for beginners

```
int x1 = 2;  
int x2 = 5;  
int sum = x1 + x2;  
System.out.println(x1 + '+' + x2 + '=' + sum);
```

How can we solve this?

Recap - Java pitfalls for beginners

Many different solutions:

1. `System.out.println(Integer.toString(x1) + '+' + x2 + '=' + sum);`
2. `System.out.println(x1 + "+" + x2 + '=' + sum);`
3. `StringBuilder sb = new StringBuilder();
sb.append(x1).append('+').append(x2).append('=')
.append(sum);
System.out.println(sb);`

...

Output now: 2+5=7

Recap - Java pitfalls for beginners

Everybody makes mistakes

Regardless of what we do, mistakes will happen, so it's best to learn how to deal with them.

For example, once you know what caused the mistake, it is important to put in place a procedure or some other sort of guard that will ensure that mistake doesn't happen again. In code, I'll often add a new unit test or some other type of automated test to ensure that a bug I fixed can never happen again.

<https://simpleprogrammer.com/2013/04/28/how-to-deal-with-making-mistakes/>

Recap - Inheritance

Example Machine continued:

```
class Machine {  
    // implicit superclass if not specified otherwise  
}  
  
public class Main {  
    public static void main(String[] args){  
        Machine machine1 = new Machine();  
        Machine machine2 = new Machine();  
        // can we do anything with the machine(s) here?  
    }  
}
```

Recap – Object(s) comparison

Two objects can be compared using operator (==) and .equals()

Recall that == tests for reference equality, not value equality. **What this means?**

Recap - Inheritance

By default (defined in `java.lang.Object`), an object is equal to another object only if it is the same instance.

But, we can provide custom equality logic when we override it.

Recap – Object(s) comparison

```
Machine machine1 = new Machine();
```

```
Machine machine2 = new Machine();
```

```
System.out.println(machine1.equals(machine2));
```

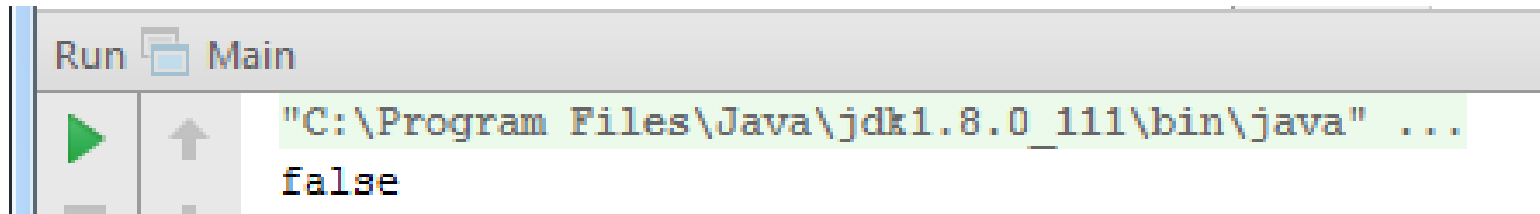
Output ?

Recap – Object(s) comparison

```
Machine machine1 = new Machine();
```

```
Machine machine2 = new Machine();
```

```
System.out.println(machine1.equals(machine2));
```



```
Run Main  
"C:\Program Files\Java\jdk1.8.0_111\bin\java" ...  
false
```

Exercise

Let's say that two instances of the Machine class are considered equal if they are of the same construction type. To test this add a String field "machineType" to the Machine class. Then create several objects (use constructor to set a construction type. i.e., provide value for a field 'machineType'), and compare them using your 'overridden' **equals()** method. After compiling you should see 'true' if you compare two Planes, and you should see 'false' if you compare Plane to a Car, for example.

Recap – Method(s) Override

```
class Machine {  
    String machineType;  
    public Machine(String machineType) {  
        this.machineType = machineType;  
    }  
    public boolean equals(Object object2) {  
        return object2 instanceof Machine &&  
            this.machineType.equals(((Machine)object2).machineType);  
    }  
}
```

```
Machine machine1 = new Machine("Plane");  
Machine machine2 = new Machine("Plane");  
Machine machine3 = new Machine("Car");  
machine1.equals(machine2) // true  
machine1.equals(machine3) // false
```

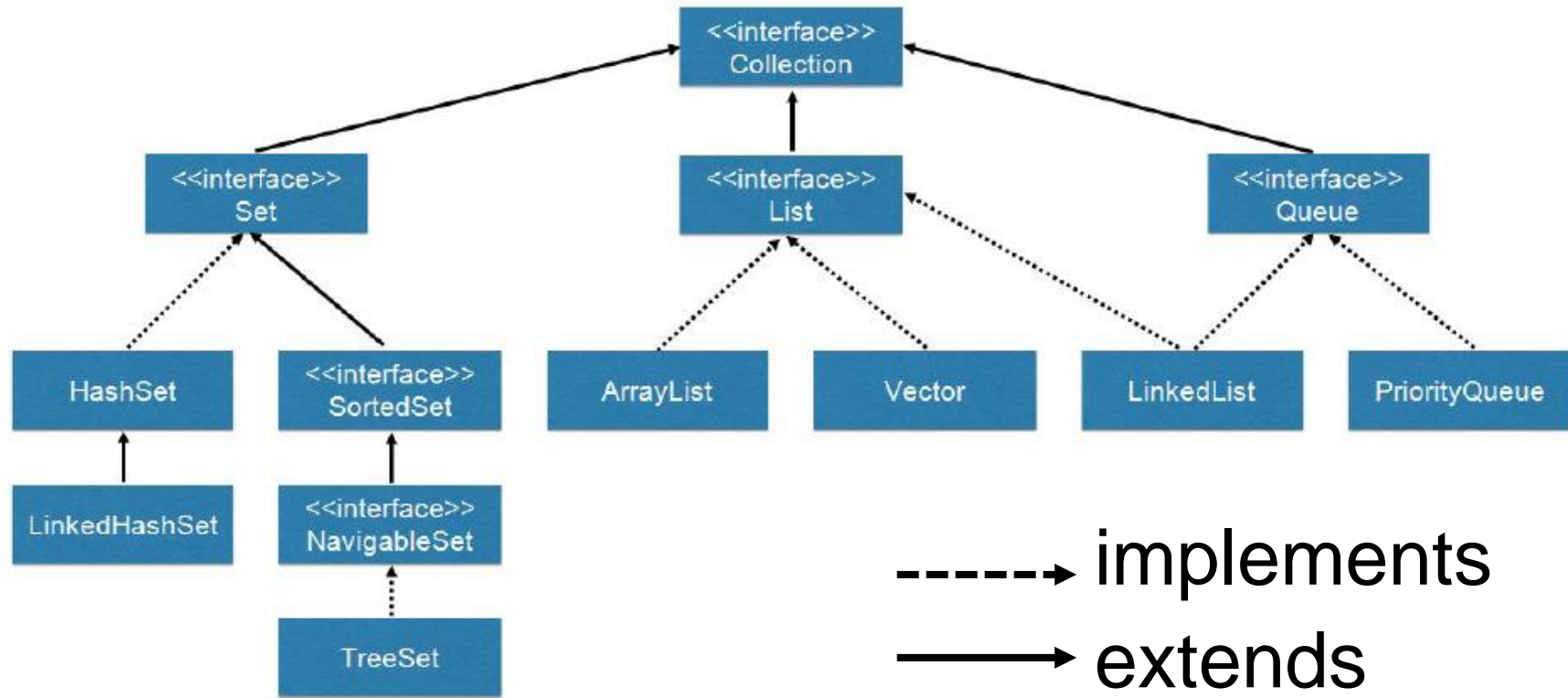
Recap – Method(s) Override

```
class Machine {  
    String machineType;  
    public Machine(String machineType) {  
        this.machineType = machineType;  
    }  
    public boolean equals(Object object2) {  
        return object2 instanceof Machine &&  
            this.machineType.equals(((Machine)object2).machineType);  
    }  
}
```

Side note: If you override equals, you almost always need to override hashCode. As it says in the equals
JavaDoc

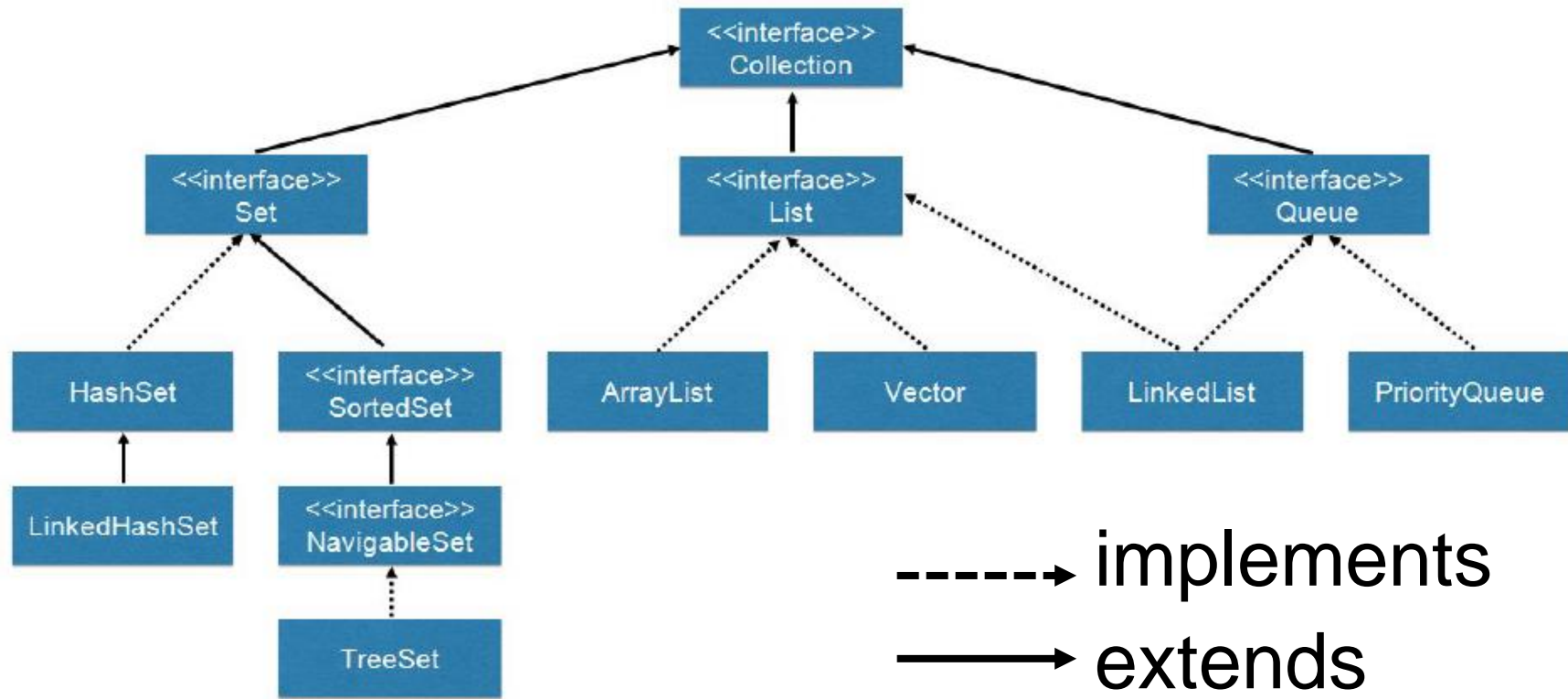
```
machine1.equals(machine2) // true  
machine1.equals(machine3) // false
```


Recap - Java Collections Framework



What are Collections ?

Recap - Java Collections Framework



A *collection* - sometimes called a container - is simply an object that groups multiple elements into a single unit.

A *collections framework* is a unified architecture for representing and manipulating collections.

Recap - Java Collections Framework

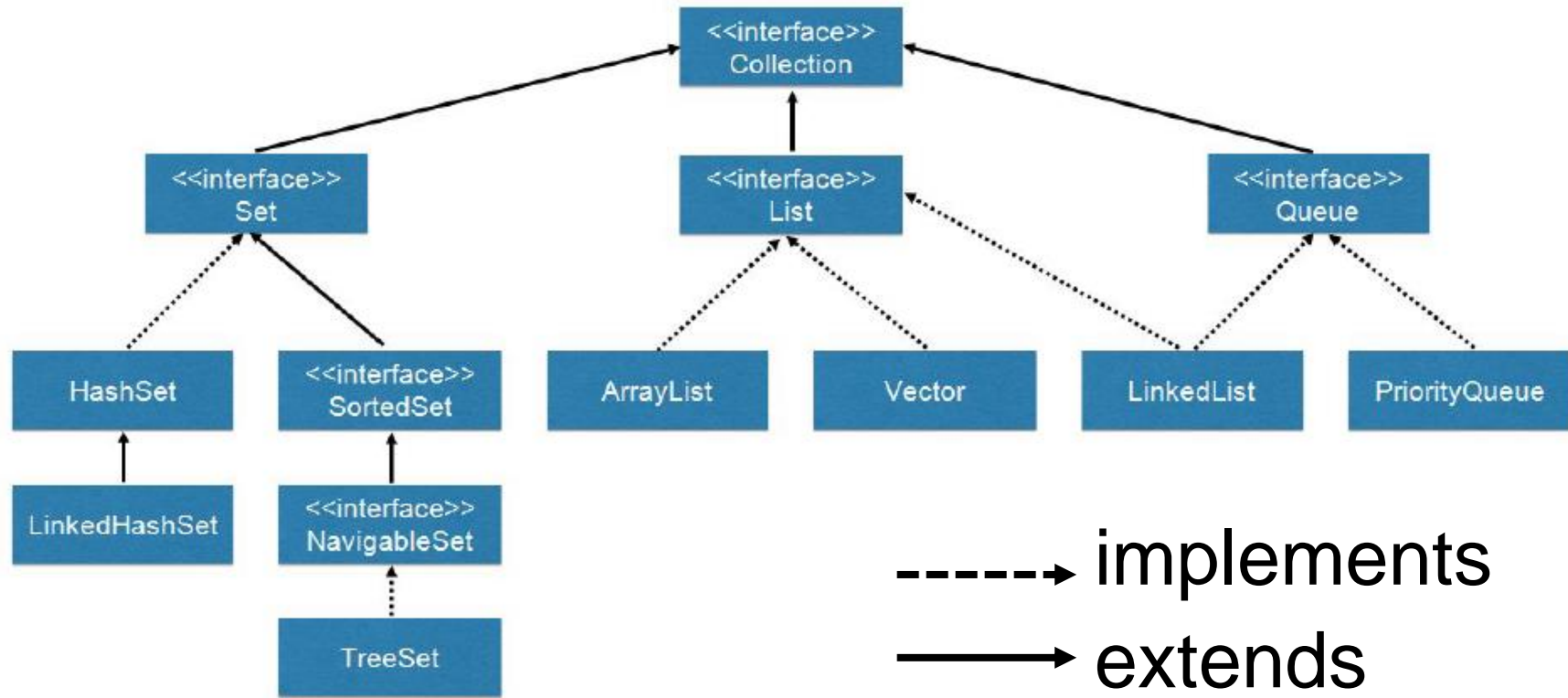
Can you name some benefits of using the Java Collections Framework?

Recap - Java Collections Framework

Can you name some benefits of using the Java Collections Framework?

- Reduces programming effort (frees you to concentrate on the important parts of your program)
- Increases program speed and quality (programs can be easily tuned by switching collection implementations)
- Fosters software reuse (standard collection interfaces are by nature reusable)
- ...

Recap - Java Collections Framework



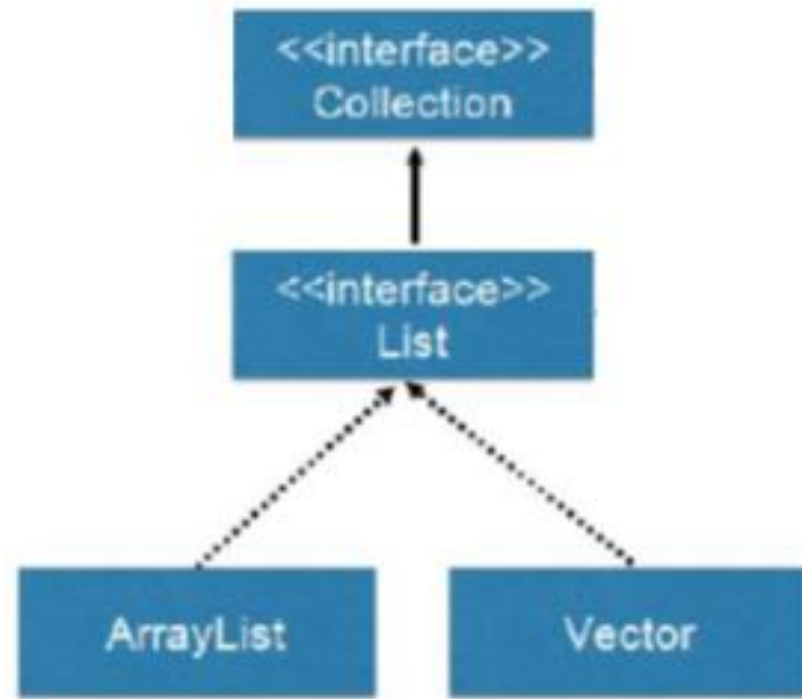
extends vs implements ?

Recap - extends vs implements

When a subclass **extends** a class, it allows the subclass to **inherit** (reuse) and **override** code defined in the supertype.

When a subclass **implements** a class, it means that it takes on the designated behavior that the interface specifies. Consider the following interface:

Recap - Collections API (List)



public interface List<E>

extends Collection<E>

public class ArrayList<E>

extends AbstractList<E>

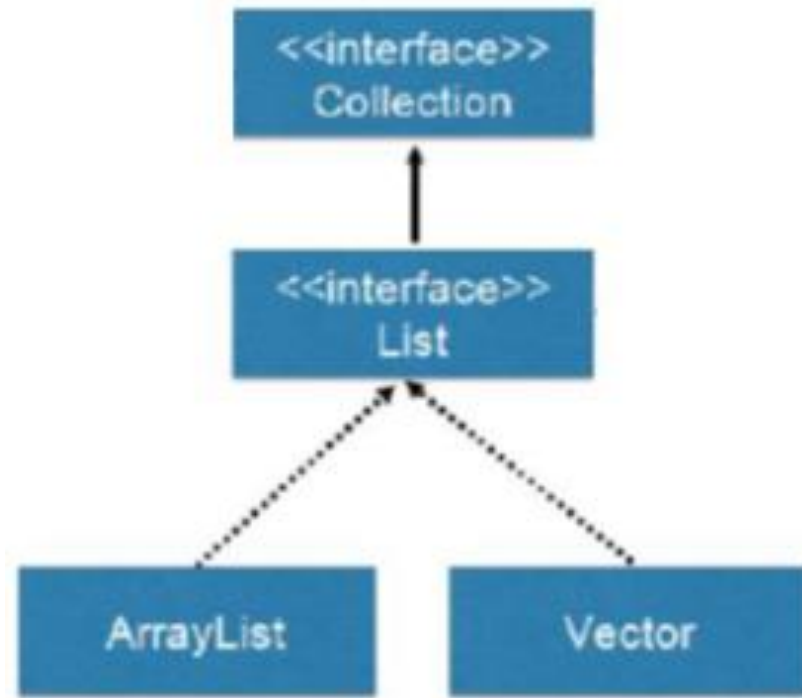
implements List<E>,

RandomAccess,
Cloneable,
Serializable

List<String> values = new ?

template type (List<E> the type of elements in this list)

Recap - Collections API (List)



public interface List<E>

extends Collection<E>

public class ArrayList<E>

extends AbstractList<E>

implements List<E>,

RandomAccess,

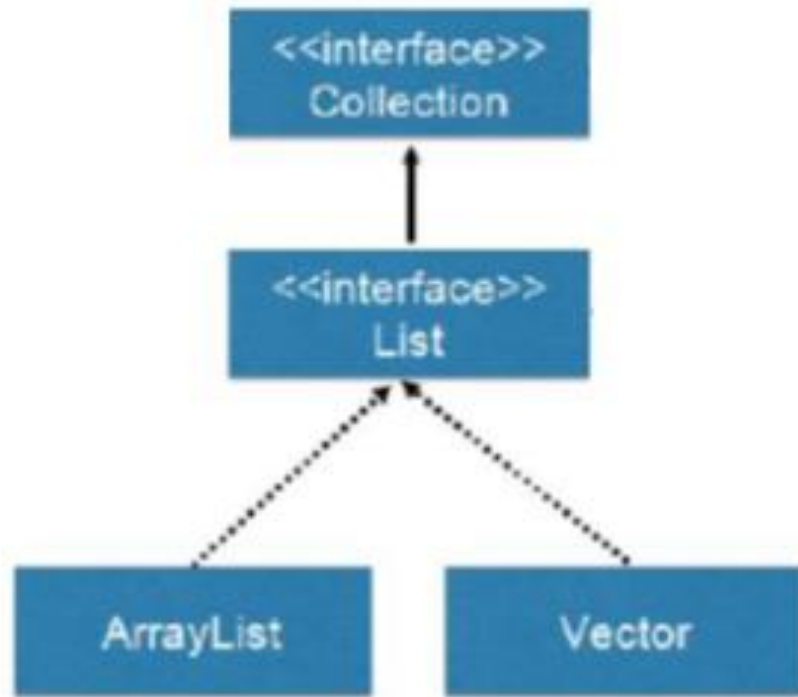
Cloneable,

Serializable

List<String> values = new List<String>();

Correct?

Recap - Collections API (Subtyping)



**<<interface>> List
cannot be instantiated**

```
List<String> values = new List<String>();  
List<String> values = new ArrayList<String>();  
List<String> values = new Vector<String>();
```

Recap - extends and implements

Both ArrayList and Vector have designated behavior that the interface specifies, for example **add** method.

```
List<String> myArrayList = new ArrayList<String>();  
myArrayList.add("A");
```

```
List<String> myVector = new Vector<String>();  
myVector.add("B");
```

Recap - extends and implements

Both ArrayList and Vector have designated behavior that the interface specifies, for example **add** method.

```
List<String> myArrayList = new ArrayList<String>();  
myArrayList.add("A");
```

▼  myArray = {ArrayList@459} size = 1
▶  0 = "A"

```
List<String> myVector = new Vector<String>();  
myVector.add("B");
```

▼  myVector = {Vector@460} size = 1
▶  0 = "B"

Recap - extends and implements

When a class **implements** an interface, it allows an object created from the class to be used in any context that expects a value of the interface.

For example, we can now refer to instances of List class through the List interface

How can we add a new value to both myArrayList and myVector using the same code?

Recap - extends and implements

When a class **implements** an interface, it allows an object created from the class to be used in any context that expects a value of the interface.

For example, we can now refer to instances of List class through the List interface

```
List<List> myDataCollection = new ArrayList<List>();
myDataCollection.add(myArrayList);
myDataCollection.add(myVector);

for (List data : myDataCollection )
{
    data.add("C");
}
```

Recap - List extends Collections

More advanced topics:

If you have Vector a and you want to copy its content exactly to Vector b, what can you do?

Recap - List extends Collections

One way to (shallow) copy elements of Vector/ArrayList is to use copy method from Collections. Or do something like this

```
List<String> b = new Vector<String>(a);
```

All elements will exist within b in the exact same order that they were within a (assuming it had an order).

Recap - List extends Collections

**How can you become a better programmer,
Code, code, code ...**