



**eC Academy**

***Realize Your Dreams***

# AP Computer Science A Review

## Week 5: Java API Classes

---

DR. ERIC CHOU  
IEEE SENIOR MEMBER

SECTION 1

# Overview



# Java API Classes for AP Exam

---

- Object Class
- Math Class
- Wrapper Classes (Integer, Double, Character)
- String Class
- Arrays Class (Non-AP)
- ArrayList Class
- List Interface
- Point2D Class (Non-AP but important)
- Scanner Class (Non-AP but important)



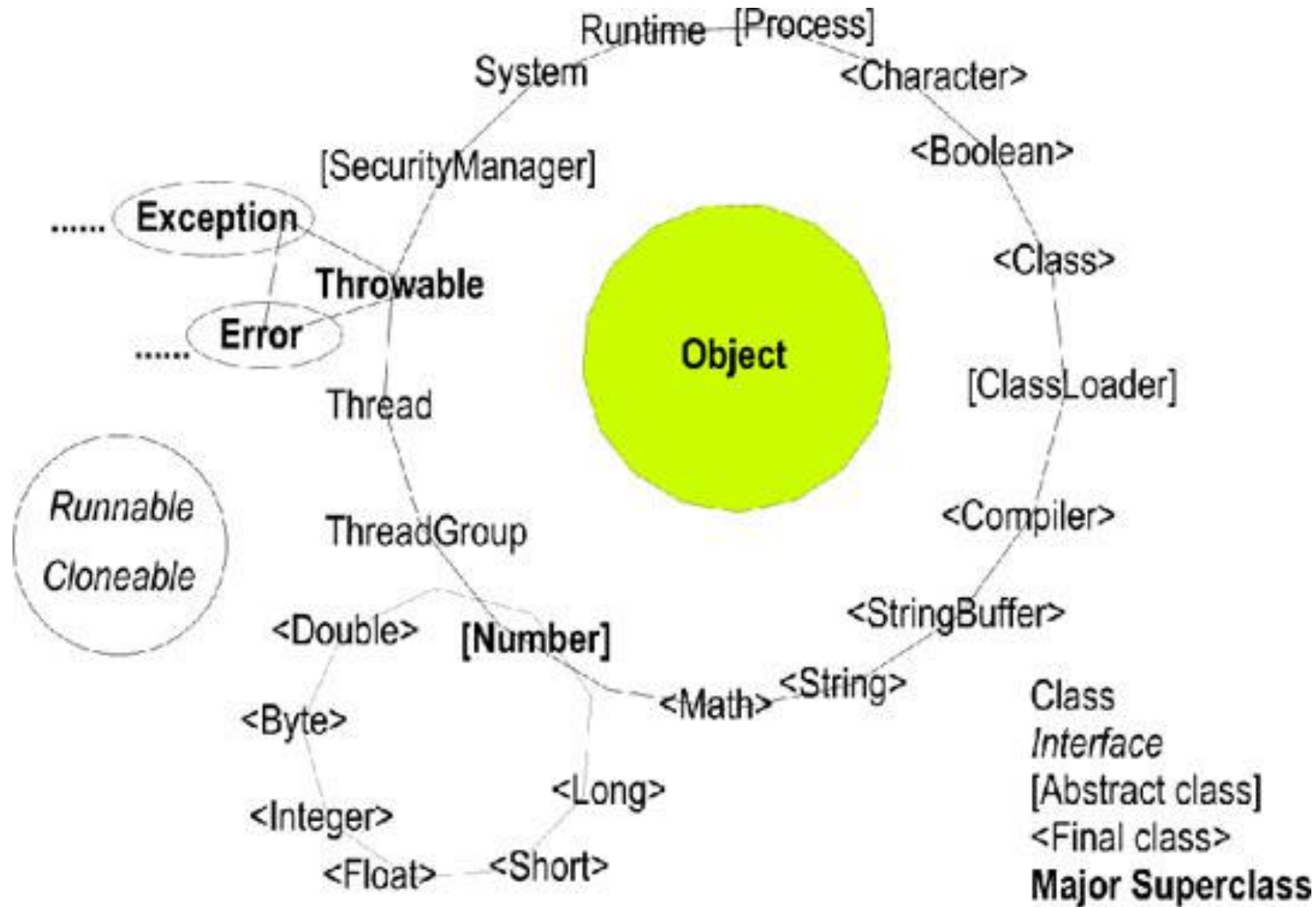
# Object Class

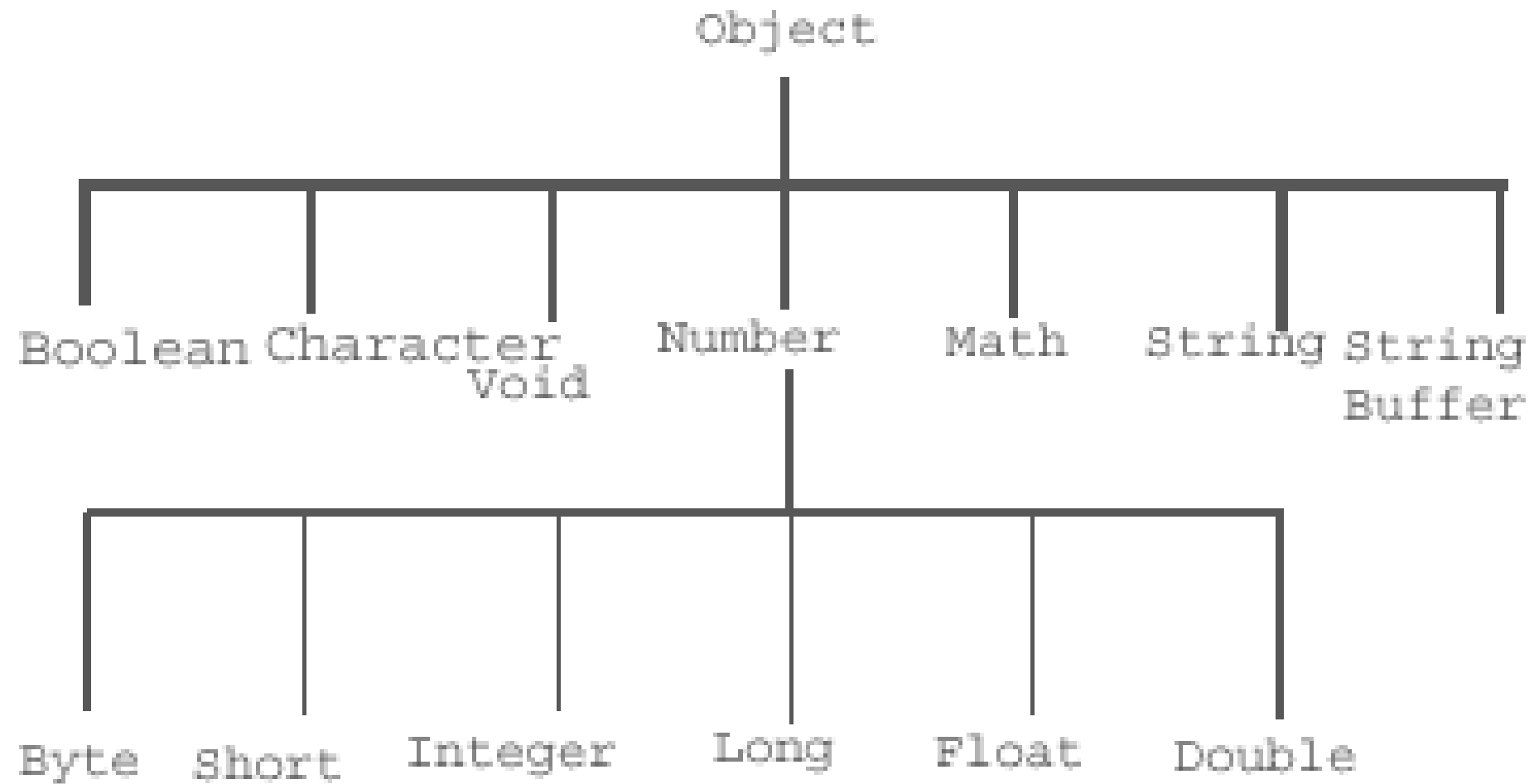
---

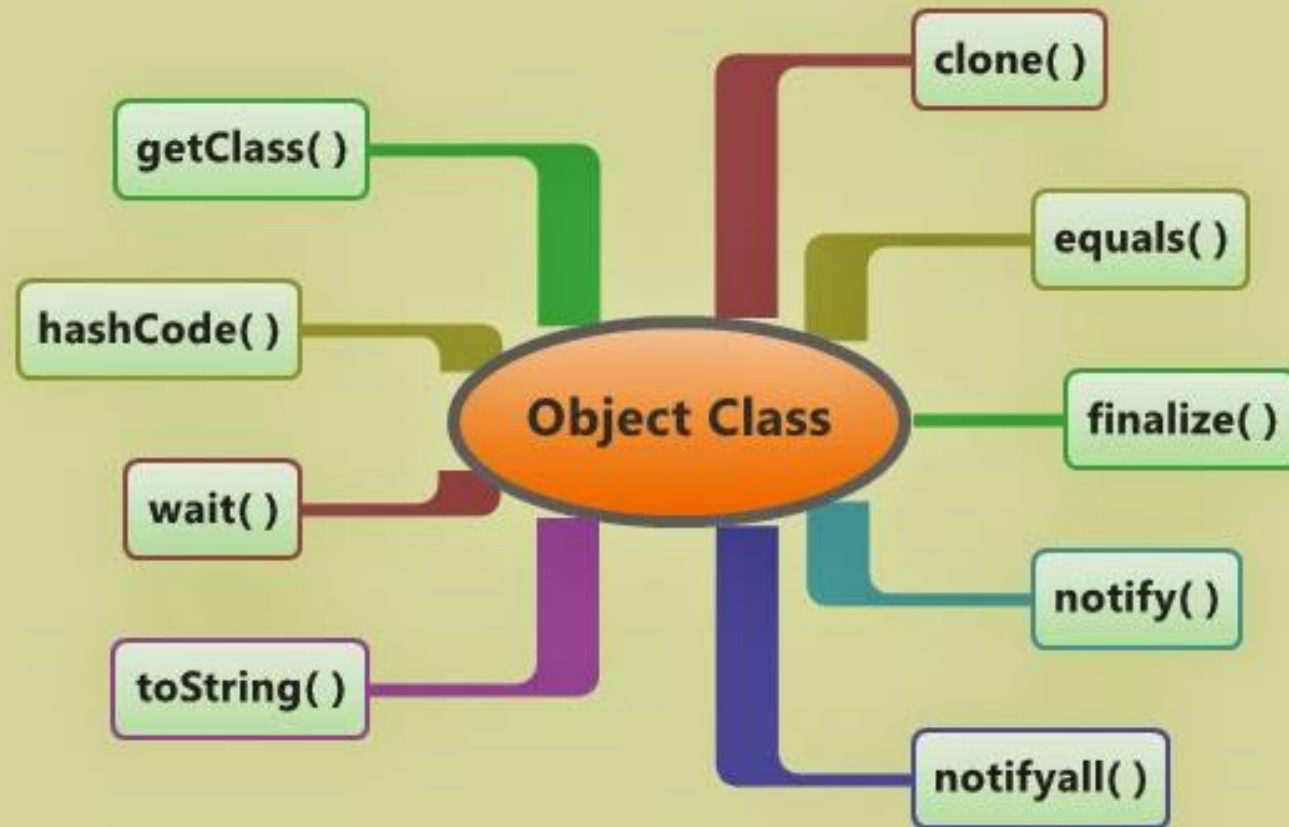
Object class is the superclass of all Java classes.

All Java classes inherited from this class. This makes it possible that we can have methods that are available in all Java classes. (Polymorphism by Inheritance)

This simplifies things compared to C++ where this is not the case.

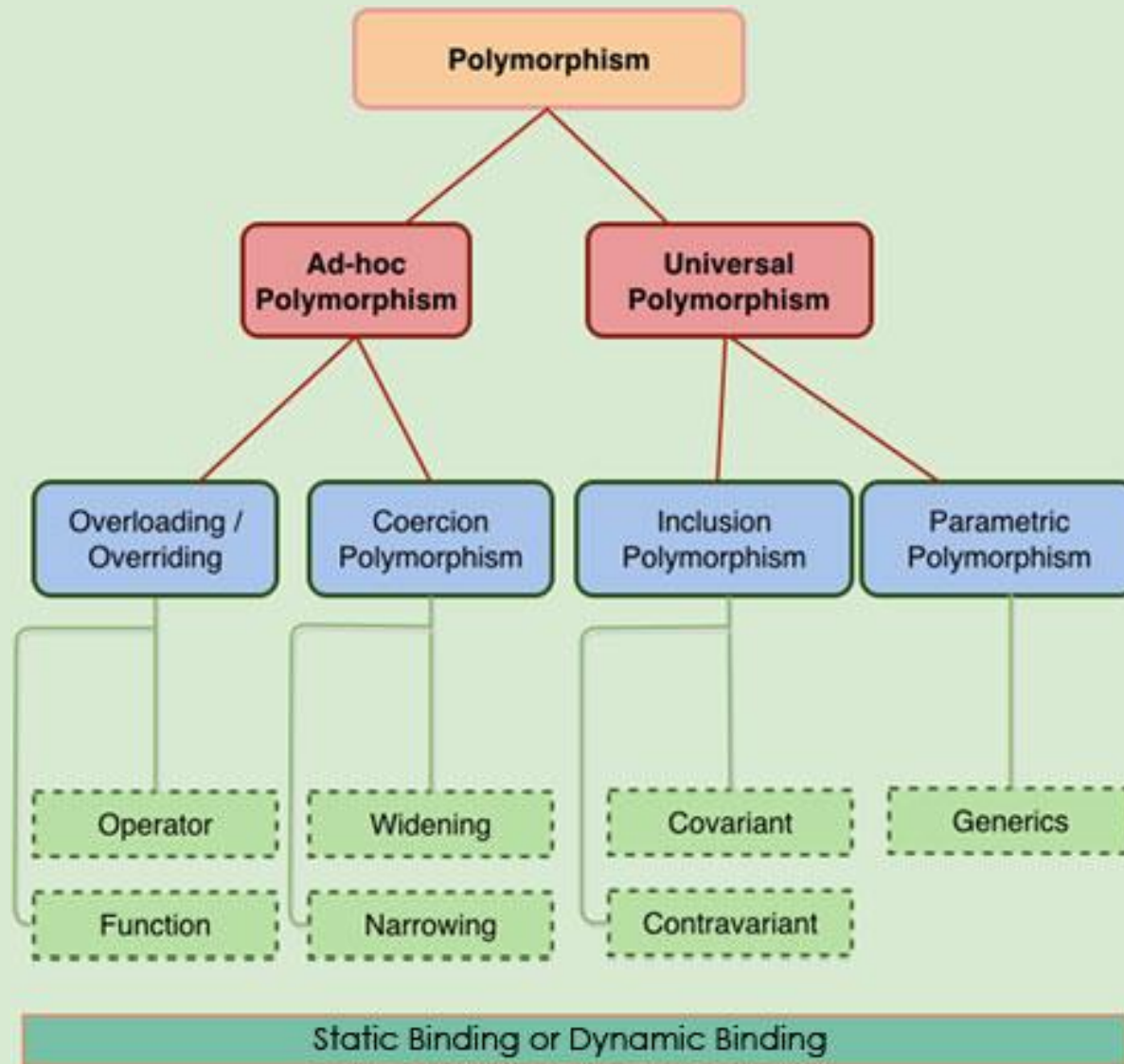






Method	Purpose
Object clone( )	Creates a new object that is the same as the object being cloned.
boolean equals(Object <i>object</i> )	Determines whether one object is equal to another.
void finalize( )	Called before an unused object is recycled.
Class<?> getClass( )	Obtains the class of an object at run time.
int hashCode( )	Returns the hash code associated with the invoking object.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( ) void wait(long <i>milliseconds</i> ) void wait(long <i>milliseconds</i> , int <i>nanoseconds</i> )	Waits on another thread of execution.





### **Overloading/Overriding:**

- Overloading: same class, different method signature
- Overriding: sub-class, same method signature

### **Coercion Polymorphism:**

- Casting: Widening/Narrowing
- Wrapper Class: auto-boxing, auto-unboxing

### **Inclusion Polymorphism:**

- Inheritance: Object Class

### **Parametric Polymorphism:**

- `ArrayList<Integer>`
- `ArrayList<E>`
- `List<E>`
- Type variable

SECTION 2

# Math Class



# Math Class

---

- Constants: `Math.PI`, `Math.E`
- Methods: `Math.pow()`, `Math.sqrt()`, `Math.max()`, `Math.min()`



# The Service Methods

Rounding Methods	Description
ceil(x)	x is rounded up to its nearest integer. This integer is returned as a double value.
floor(x)	x is rounded down to its nearest integer. This integer is returned as a double value.
rint(x)	x is rounded up to its nearest integer. If x is equally close to two integers, the even one is returned as a double value
round(x)	Returns (int) Math.floor(x+0.5) if x is a float and returns (long) Math.floor(x+0.5) if x is a double.
Min Max Abs Methods	Description
max(x, y)	Return the greater number between x and y.
min(x, y)	Return the less number between x and y
abs(x)	Return the absolute value of x
Random Methods	Description
random()	Return a random number between $0 \leq y < 1$ . (Compared to java.util.Random)

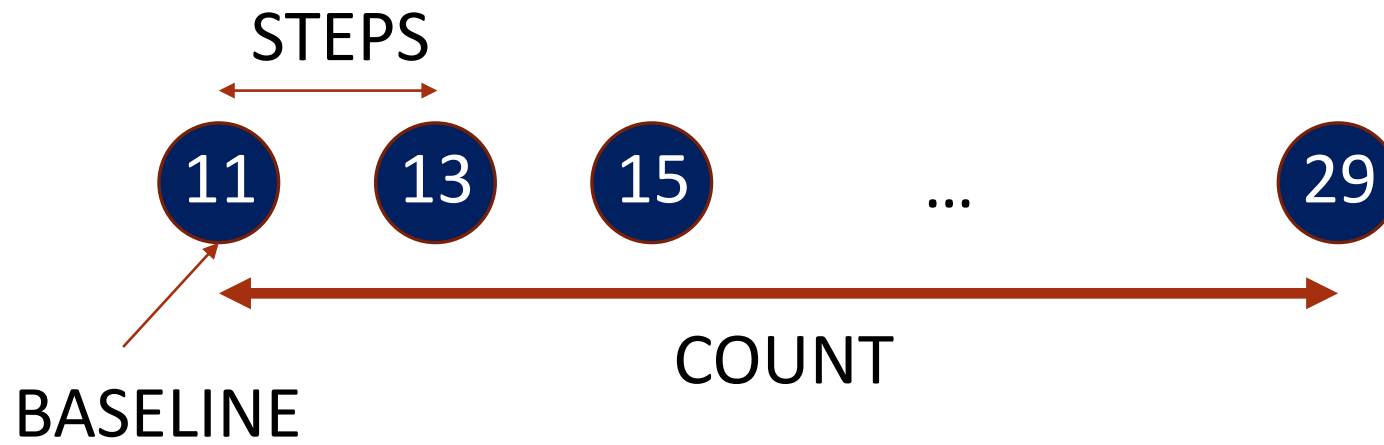
# The Exponent Methods

Methods	Description	
<code>exp(x)</code>	Returns e raised to power of x	$(e^x)$
<code>log(x)</code>	Return the natural logarithm of x	$(\ln(x) = \log_e(x))$ .
<code>log10(x)</code>	Returns the base 10 algorithm of x	$(\log_{10}(x))$ .
<code>pow(a, b)</code>	Returns a raised to the power of b	$(a^b)$
<code>sqrt(x)</code>	Returns the square root of x	$(\sqrt{x})$ for $x \geq 0$

# Review of Random Sample Generation

```
int r = ((int) (Math.random()*COUNT) * STEPS + BASELINE;
```

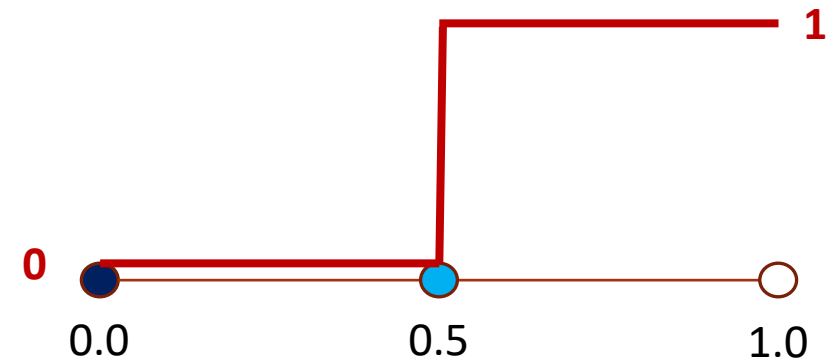
```
int r = ((int) (Math.random() * 10) * 2 + 11);
```



# Un-biased Randomized Coin (50-50)

Unbiased Random Number Generator:

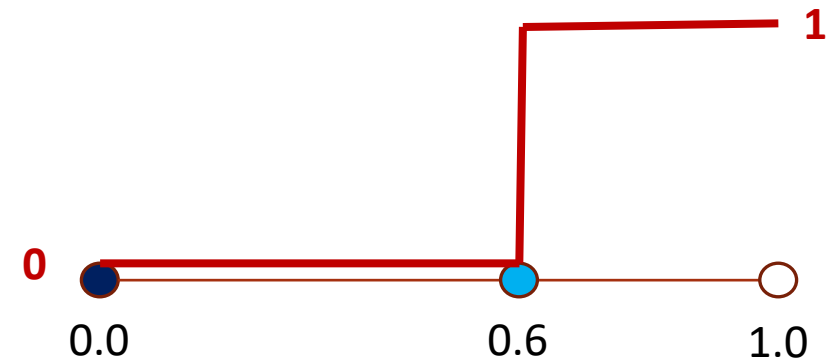
```
double randToss = Math.random();
int die = 1;
if (randToss <= 0.5) die = 0; // preset-else
// Think about it, you do not need the else-part.
// another way to write it.
int ide = (randToss<=0.5) ? 0 : 1;
// conditional expression.
//(coming lecture in this chapter)
```



# Biased Randomized Coin (60% - 0 (Tail))

Unbiased Random Number Generator:

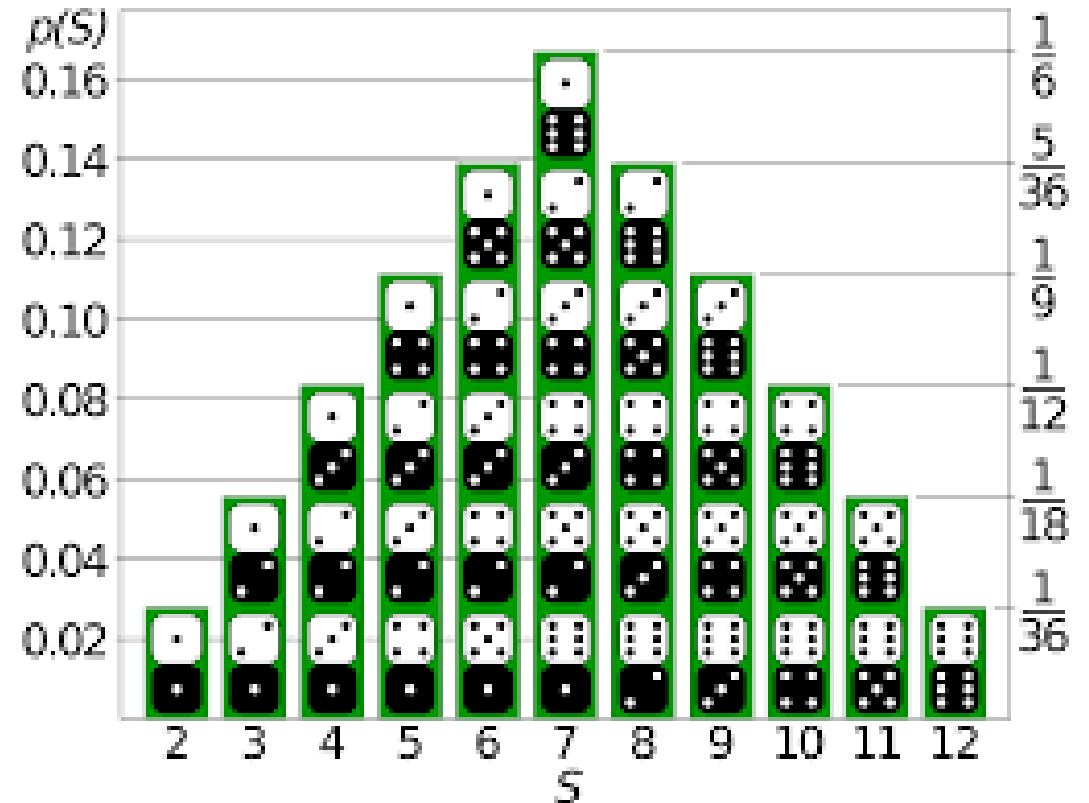
```
double randToss = Math.random();  
int die = 1;  
if (randToss < 0.6) die = 0; // preset-else  
// Think about it, you do not need the else-part.  
// another way to write it.  
int ide = (randToss <= 0.6) ? 0 : 1;  
// conditional expression.  
//(coming lecture in this chapter)
```





# Sum of Two Dice Randomized Test

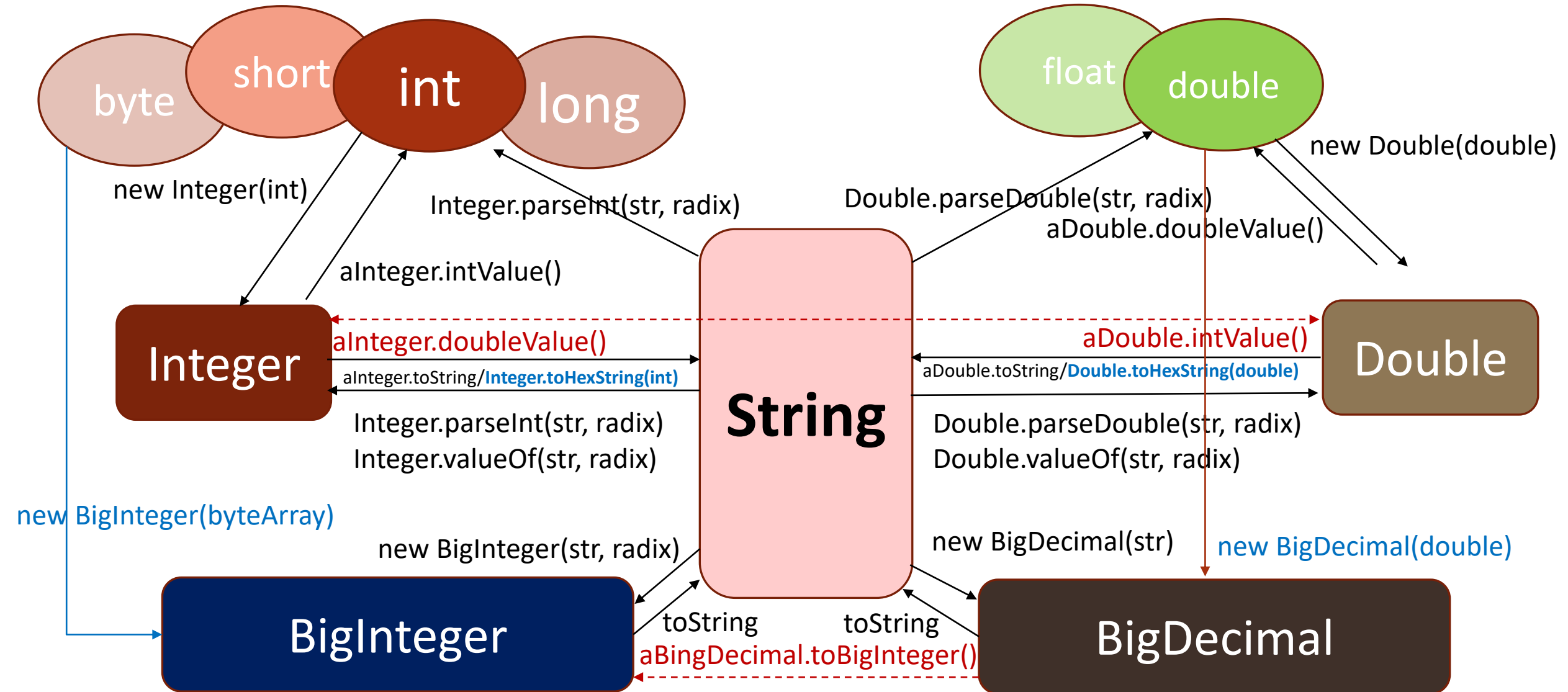
```
int die1 = (int) (Math.random()*6) + 1;
int die2 = (int) (Math.random()*6) + 1;
int sum = die1 + die2;
```



## SECTION 3

# Wrapper Classes

# Map for Java Number Space



# The Integer and Double Classes

<b>java.lang.Integer</b>
-value: int +MAX VALUE: <u>int</u> +MIN VALUE: <u>int</u>
+Integer(value: int) +Integer(s: String) +byteValue(): byte +shortValue(): short +intValue(): int +longVlaue(): long +floatValue(): float +doubleValue(): double +compareTo(o: Integer): int +toString(): String +valueOf(s: String): <u>Integer</u> +valueOf(s: String, radix: int): <u>Integer</u> + <u>parseInt(s: String): int</u> + <u>parseInt(s: String, radix: int): int</u>

<b>java.lang.Double</b>
-value: double +MAX VALUE: <u>double</u> +MIN VALUE: <u>double</u>
+Double(value: double) +Double(s: String) +byteValue(): byte +shortValue(): short +intValue(): int +longVlaue(): long +floatValue(): float +doubleValue(): double +compareTo(o: Double): int +toString(): String +valueOf(s: String): <u>Double</u> +valueOf(s: String, radix: int): <u>Double</u> + <u>parseDouble(s: String): double</u> + <u>parseDouble(s: String, radix: int): double</u>



# Numeric Wrapper Class Constants

---

Each numerical wrapper class has the constants MAX\_VALUE and MIN\_VALUE. MAX\_VALUE represents the maximum value of the corresponding primitive data type.

For Byte, Short, Integer, and Long, MIN\_VALUE represents the minimum byte, short, int, and long values. For Float and Double, MIN\_VALUE represents the minimum *positive* float and double values.

The following statements display the maximum integer (2,147,483,647), the minimum positive float (1.4E-45), and the maximum double floating-point number (1.79769313486231570e+308d).



# Conversion Methods

---

Each numeric wrapper class implements the abstract methods doubleValue, floatValue, intValue, longValue, and shortValue, which are defined in the Number class.

These methods “convert” objects into primitive type values.



## The Static valueOf Methods

---

The numeric wrapper classes have a useful class method, `valueOf(String s)`. This method creates a new object initialized to the value represented by the specified string. For example:

```
Double doubleObject = Double.valueOf("12.4");
```

```
Integer integerObject = Integer.valueOf("12");
```



# The Methods for Parsing Strings into Numbers

---

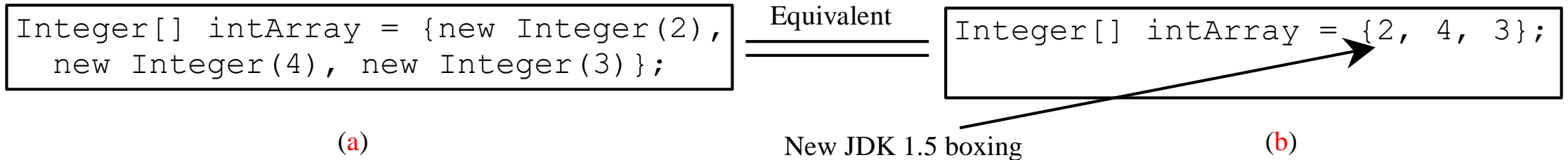
You have used the `parseInt` method in the `Integer` class to parse a numeric string into an `int` value and the `parseDouble` method in the `Double` class to parse a numeric string into a `double` value.

Each numeric wrapper class has two overloaded parsing methods to parse a numeric string into an appropriate numeric value.

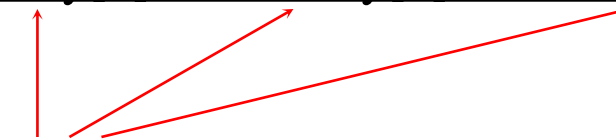


# Automatic Conversion Between Primitive Types and Wrapper Class Types

JDK 1.5 allows primitive type and wrapper classes to be converted automatically. For example, the following statement in (a) can be simplified as in (b):



```
Integer[] intArray = {1, 2, 3};  
System.out.println(intArray[0] + intArray[1] + intArray[2]);
```

  
 Unboxing



## FindMax (Two Styles)

---

- The first style preset the maximum to `int max = Integer.MIN_VALUE`
- The second style preset the maximum to `int max = array[0];`

```

1 public class FindMax{
2     static int[] ary = {3, 5, 8, 12, 4, 0, 1, -4, 9, 17};
3
4
5     public static int findMax1(int[] ary){
6         int max = Integer.MIN_VALUE;
7         for (int i=0; i<ary.length; i++){
8             if (ary[i] >= max) max = ary[i];
9         }
10        return max;
11    }
12
13    public static int findMax2(int[] ary){
14        int max;
15        if (ary.length>0) max = ary[0]; else return Integer.MIN_VALUE;
16        for (int i=1; i<ary.length; i++){
17            if (ary[i] >= max) max = ary[i];
18        }
19        return max;
20    }
21
22    public static void main(String[] args){
23        System.out.println(findMax1(ary));
24        System.out.println(findMax2(ary));
25    }
26 }

```

BlueJ: Terminal Window - Week2

Options

17

17

## SECTION 4

# String Classes



# The String Class

---

- Constructing a String:
  - `String message = "Welcome to Java";`
  - `String message = new String("Welcome to Java");`
  - `String s = new String();`
- Obtaining String length and Retrieving Individual Characters in a string
- String Concatenation (concat)
- String Conversions



# The String Type

A string is a sequence of characters.

---

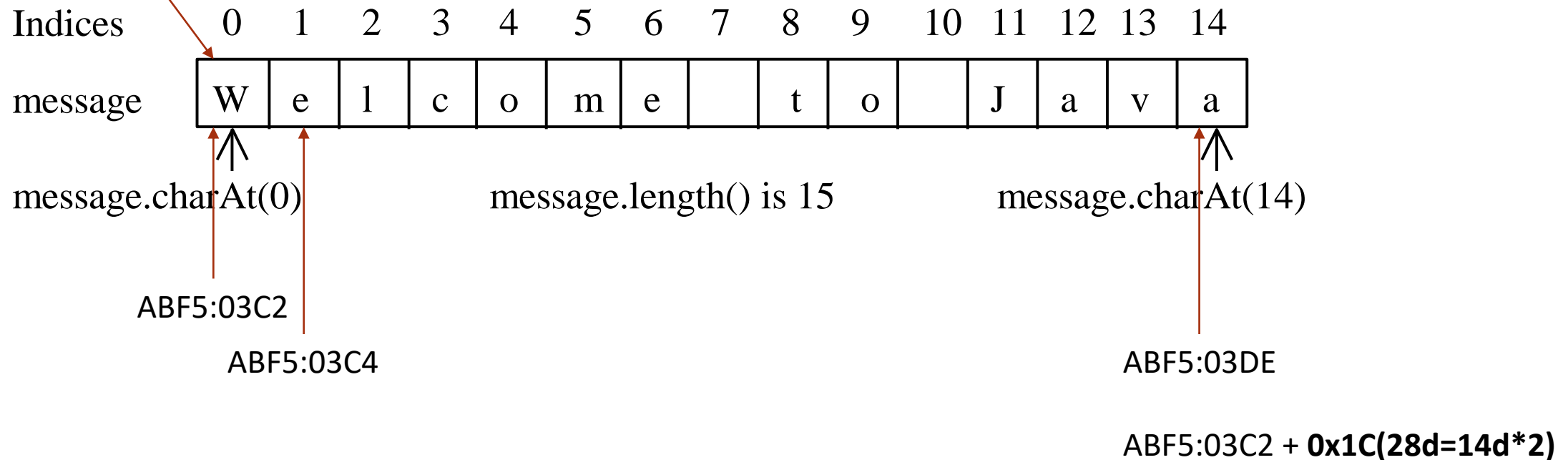
The char type represents only one character. To represent a string of characters, use the data type called **String**.

**String** is a predefined class in the Java library, just like the classes **System** and **Scanner**. The **String** type is not a primitive type. It is known as a *reference type*.

Method	Description
length()	Returns the number of characters in this string.
charAt(index)	Returns the character at the specified index from this string.
concat(s1)	Returns a new string that concatenates this string with string s1.
toUpperCase()	Returns a new string with all letters in uppercase.
toLowerCase()	Returns a new string with all letters in lowercase
trim()	Returns a new string with whitespace characters trimmed on both sides.

# Reference Data Type (only a pointer)

String message = "Welcome to Java"; // message does not store the data  
// but store address ABF5:03C2





# Strings Are Immutable

---

A String object is immutable; its contents cannot be changed. Does the following code change the contents of the string?

```
String s = "Java";
```

```
s = "HTML";
```

- s is only a reference (pointer).



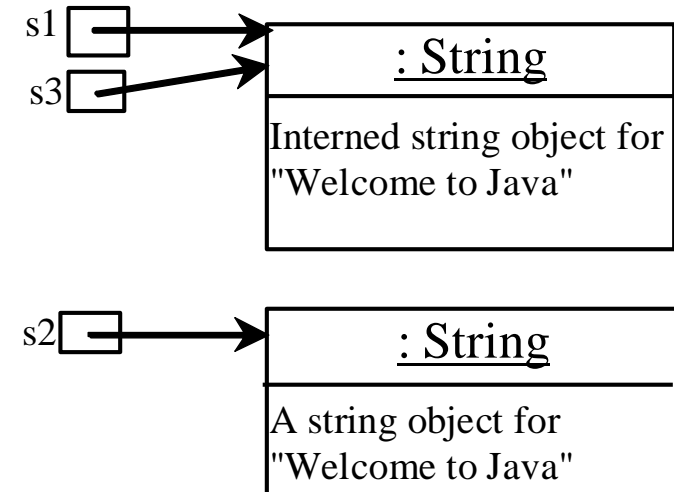
# Examples

```
String s1 = "Welcome to Java";
```

```
String s2 = new String("Welcome to Java");
```

```
String s3 = "Welcome to Java";
```

```
System.out.println("s1 == s2 is " + (s1 == s2));  
System.out.println("s1 == s3 is " + (s1 == s3));
```



display

s1 == s2 is false

s1 == s3 is true

A new object is created if you use the new operator.

If you use the string initializer, no new object is created if the interned object is already created.

# Concatenating Strings

---

1. By concat() function,

```
String s3 = s1.concat(s2);
```

2. By addition,

```
String s3 = s1 + s2;
```

3. By increment assignment,

```
message += "and Java is fun";
```



# Converting Strings

---

## Lowercase:

`"Welcome".toLowerCase()` -> `"welcome"`

## Uppercase:

`"Welcome".toUpperCase()` -> `"WELCOME"`

**Trim:** get rid of ' ', \t (tab), \f (new page), \r (carriage retron), \n (new line)(whitespace letters)

`"\t Good Night \n".trim()` -> `"Good Night"`



# Comparing Strings

---

Method	Description
<code>equals(s1)</code>	Returns true if this string is equal to string s1.
<code>equalsIgnoreCase(s1)</code>	Returns true if this string is equal to string s1; it is case insensitive.
<code>compareTo(s1)</code>	Returns an integer greater than 0, equal to 0, or less than 0 to indicate whether this string is greater than equal to or less than s1.
<code>compareToIgnoreCase(s1)</code>	Same as <code>compareTo</code> except that the comparison is case insensitive.
<code>startsWith(prefix)</code>	Returns true if this string starts with the specified prefix.
<code>endsWith(suffix)</code>	Returns true if this string ends with the specified suffix.
<code>contains(s1)</code>	Returns if s1 is a substring in this string.

# String Comparisons

---

**equals (Object object) :**

```
String s1 = new String("Welcome");
```

```
String s2 = "welcome";
```

```
if (s1.equals(s2)) { // false
```

```
    // if s1 and s2 have the same contents
```

```
}
```

```
if (s1 == s2) { // false
```

```
    // if s1 and s2 have the same reference
```

```
}
```

## String Comparisons, cont.

---

### **compareTo (Object object) :**

```
String s1 = new String("Welcome");  
String s2 = "welcome";  
    if (s1.compareTo(s2) > 0) {  
        // s1 is greater than s2  
    }  
    else if (s1.compareTo(s2) == 0) {  
        // s1 and s2 have the same contents  
    }  
    else {  
        // s1 is less than s2  
    }
```



# Finding a Character or a Substring in a String

Method	Description
indexOf(ch)	Returns the index of the first occurrence of ch in the string. Returns -1 if not matched.
indexOf(ch, fromIndex)	Returns the index of the first occurrence of ch <b>after</b> fromIndex in the string. Returns -1 if not matched.
indexOf(s)	Returns the first occurrence of string s in this string. Returns -1 if not matched
indexOf(s, fromIndex)	Returns the index of the first occurrence of string s in this string <b>after</b> fromIndex. Returns -1 if not matched.
lastIndexOf(ch)	Returns the index of the last occurrence of ch in the string . Returns -1 if not matched.
lastIndexOf(ch,fromIndex)	Returns the index of the last occurrence of ch <b>before</b> fromIndex in this string. Returns -1 if not matched.
lastIndexOf(s)	Returns the index of the last occurrence of string s. Returns -1 if not matched.
lastIndexOf(s, fromIndex)	Returns the index of the last occurrence of string s <b>before</b> fromIndex -1 if not matched.

# Finding a Character or a Substring in a String

---

`"Welcome to Java".indexOf('W')` returns 0.

`"Welcome to Java".indexOf('x')` returns -1.

`"Welcome to Java".indexOf('o', 5)` returns 9.

`"Welcome to Java".indexOf("come")` returns 3.

`"Welcome to Java".indexOf("Java", 5)` returns 11.

`"Welcome to Java".indexOf("java", 5)` returns -1.

`"Welcome to Java".lastIndexOf('a')` returns 14.





# Demo Program: AllOccurrence.java

---

- `public ArrayList<Integer> allOccurrence(String source, String pattern);`

**source:** string to be scanned

**pattern:** pattern which we would like to find all of its occurrence  
(non-overlapping)

**return:** ArrayList of all the indice value for occurrences of the pattern in  
the source string.

**Example:** `allOccurrence(" Ha !", "Ha")` will return `[0, 3, 6]`

If the arraylist returned has a size of 0, that means there is no  
pattern in the source string.

```

1 import java.util.ArrayList;
2 public class AllOccurrence{
3     public static ArrayList<Integer> allOccurrence(String s, String p){
4         ArrayList<Integer> ilist = new ArrayList<Integer>();
5         for (int i=0; i<s.length(); ){
6             int index = s.indexOf(p, i);
7             if (index>=0){
8                 ilist.add(index);
9                 i = index + p.length();
10            }
11            else i=s.length();
12        }
13        return ilist;
14    }
15
16    public static void main(String[] args){
17        System.out.print("\f");
18        String source = "Old MACDONALD had a farm. "+
19        "E-I-E-I-O. And on his farm he had a cow. "+
20        "E-I-E-I-O. With a moo moo here. And a moo moo there"+
21        "Here a moo, there a moo. Everywhere a moo moo. "+
22        "Old MacDonald had a farm. E-I-E-I-O";
23        ArrayList<Integer> ilist = allOccurrence(source, "moo");
24        System.out.println("Occurrence of moo="+ilist);
25        ilist = allOccurrence(source, "E-I-E-I-O");
26        System.out.println("Occurrence of E-I-E-I-O="+ilist);
27    }
28 }

```

Blue: Terminal Window - Week2

Options

Occurrence of moo=[85, 89, 105, 109, 125, 138, 156, 160]  
 Occurrence of E-I-E-I-O=[26, 67, 191]

# Converting, Replacing and Splitting



## java.lang.String

+toLowerCase(): String

Returns a new string with all characters converted to lowercase.

+toUpperCase(): String

Returns a new string with all characters converted to uppercase.

+trim(): String

Returns a new string with blank characters trimmed on both sides.

+replace(oldChar: char,  
newChar: char): String

Returns a new string that replaces all matching character in this string with the new character.

+replaceFirst(oldString: String,  
newString: String): String

Returns a new string that replaces the first matching substring in this string with the new substring.

+replaceAll(oldString: String,  
newString: String): String

Returns a new string that replace all matching substrings in this string with the new substring.

+split(delimiter: String):  
String[]

Returns an array of strings consisting of the substrings split by the delimiter.

## Examples

"Welcome".toLowerCase() returns a new string, welcome.

"Welcome".toUpperCase() returns a new string, WELCOME.

" Welcome ".trim() returns a new string, Welcome.

"Welcome".replace('e', 'A') returns a new string, WAlcomA.

"Welcome".replaceFirst("e", "AB") returns a new string, WABlcome.

"Welcome".replace("e", "AB") returns a new string, WABlcomAB.

"Welcome".replace("el", "AB") returns a new string, WABcome.



# Escape Sequences for Special Characters

Escape Sequences			
Escape Sequence	Name	Unicode Code	Decimal Value
\b	Backspace	\u0008	8
\t	Tab	\u0009	9
\n	Linefeed	\u000A	10
\f	Formfeed	\u000C	12
\r	Carriage Return	\u000D	13
\\	Backslash	\u005C	92
\"	Double Quote	\u0022	34



# Formatting Console Output

**System.out.printf:** You can use the System.out.printf method to display formatted output on the console.

Formatted out provides programmer to output data in a specified data type and data precision.

Eg. `System.out.printf("Interest is $%4.2f", 12618.98*0.0013);`

Output is: Interest is \$16.40

Format Specifier	Output	Example
%b	A Boolean value	True or false
%c	A character	'a'
%d	A decimal integer	200
%f	A floating-point number	45.460000
%e	A number in standard scientific notation	4.556000e+01
%s	A string	"Java is cool"
%n	A newline mark	Like "\n" but better.
%tc	complete date and time	
%%	The character %	

# Specifying Width and Precision

Example	Output
%5c	Output the character and add four spaces before the character item, because the width is 5.
%6b	Output the Boolean value and add one space before the false value and two spaces before the true value.
%5d	Output the integer item with width at least 5. If the number of digits in the item is < 5, add spaces before the number. If the number of digits in the item is > 5, the width is automatically increased.
%10.2f	Output the floating-point item with width at least 10 including a decimal point and two digits after the point. Thus, there are 7 digits allocated before the decimal point. If the number of digits before the decimal point in the item is < 7, add spaces > 7, the width is automatically increased.
%10.2e	Output the floating-point item with width at least 19 including a decimal point, two digits after the point and the exponent part. If the displayed number in scientific notation has width less than 10, add spaces before the number.
%12s	Output the string with width at least 12 characters. If the string item has fewer than 12 characters, add spaces before the string. If the string item has more than 12 characters, the width is automatically increased.



# printf integer formatting

---

As a summary of printf integer formatting, here's a little collection of integer formatting examples. Several different options are shown, including a minimum width specification, left-justified, zero-filled, and also a plus sign for positive numbers.

Description	Code	Result
At least five wide	<code>System.out.printf("%5d", 10);</code>	' 10 '
At least five-wide, left-justified	<code>System.out.printf("%-5d", 10);</code>	'10  '
At least five-wide, zero-filled	<code>System.out.printf("%05d", 10);</code>	'00010 '
At least five-wide, with a plus sign	<code>System.out.printf("%+5d", 10);</code>	' +10 '
Five-wide, plus sign, left-justified	<code>System.out.printf("%-+5d", 10);</code>	' +10  '



# printf - floating point numbers

Here are several examples showing how to format floating-point numbers with printf:



Description	Code	Result
Print one position after the decimal	<code>System.out.printf("%.1f", 10.3456);</code>	<code>'10.3'</code>
Two positions after the decimal	<code>System.out.printf("%.2f", 10.3456);</code>	<code>'10.35'</code>
Eight-wide, two positions after the decimal	<code>System.out.printf("%8.2f", 10.3456);</code>	<code>' 10.35'</code>
Eight-wide, four positions after the decimal	<code>System.out.printf("%8.4f", 10.3456);</code>	<code>' 10.3456'</code>
Eight-wide, two positions after the decimal, zero-filled	<code>System.out.printf("%08.2f", 10.3456);</code>	<code>'00010.35'</code>
Eight-wide, two positions after the decimal, left-justified	<code>System.out.printf("%-8.2f", 10.3456);</code>	<code>'10.35 '</code>
Printing a much larger number with that same format	<code>System.out.printf("%-8.2f", 101234567.3456);</code>	<code>'101234567.35'</code>

SECTION 5

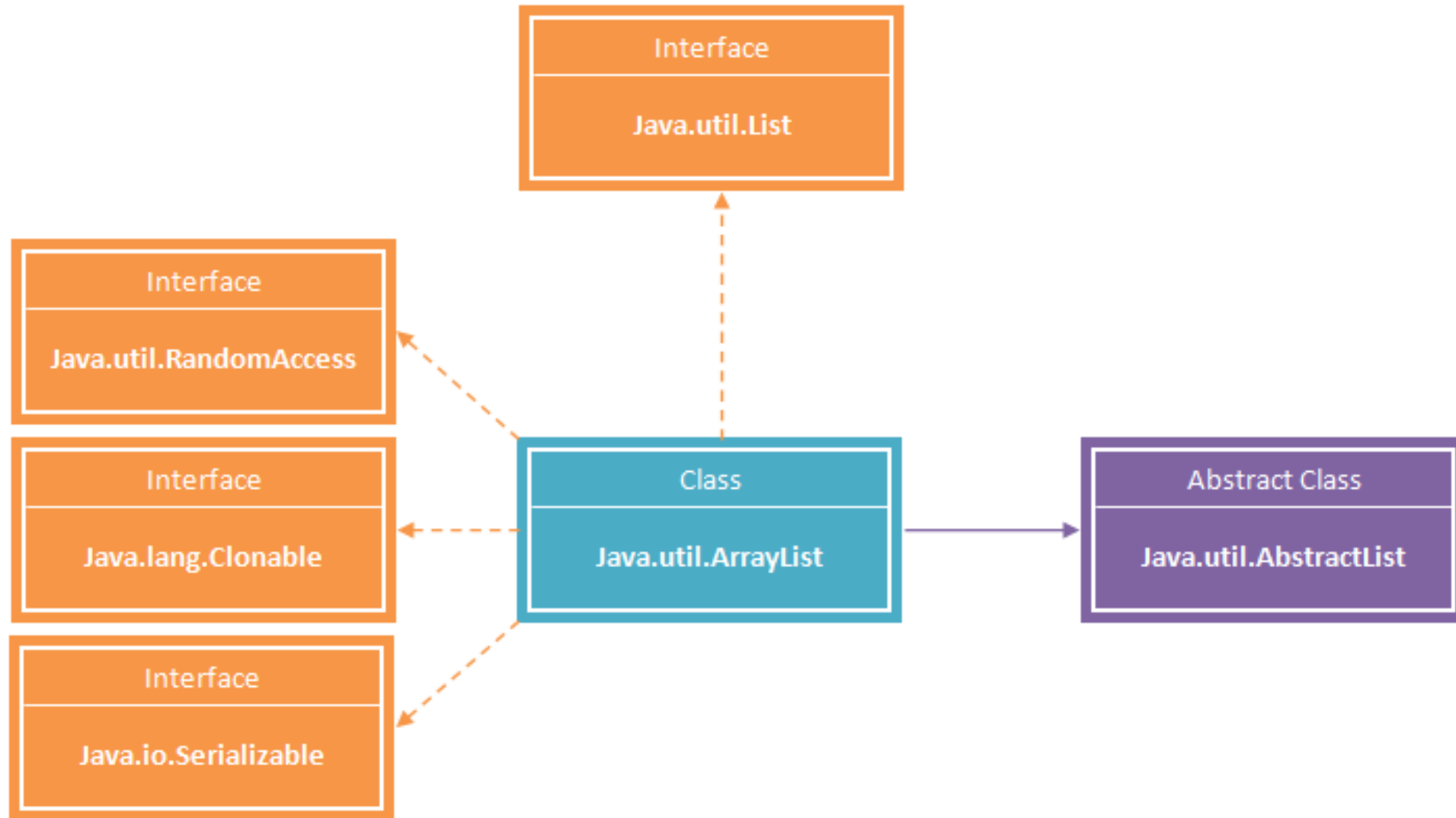
# Arrays and ArrayList Classes



# Arrays Class (Non-AP)

---

- Class Arrays helps you avoid reinventing the wheel by providing static methods for common array manipulations
- Methods include
  - **sort(array)**: Arranges array elements into increasing order (Ascending).
  - **binarySearch(array, element)**: Determines whether an array contains a specific value and, if so, returns where the value located.
  - **equals(array)**: Compares arrays.
  - **fill(array, element)**: Places Values into an array.
  - **toString()**: Converts array to String
  - **asList(array)**: Convert an array into a list (arraylist).



## Methods in ArrayList

---

- `boolean add(Object e)`
- `void add(int index, Object element)`
- `boolean addAll(Collection c)`
- `Object get(int index)`
- `Object set(int index, Object element)`
- `Object remove(int index)`
- `Iterator iterator()`
- `ListIterator listIterator()`
- `int indexOf()`
- `int lastIndexOf()`
- `int index(Object element)`
- `int size()`
- `void clear()`

# Comparison between Array and String



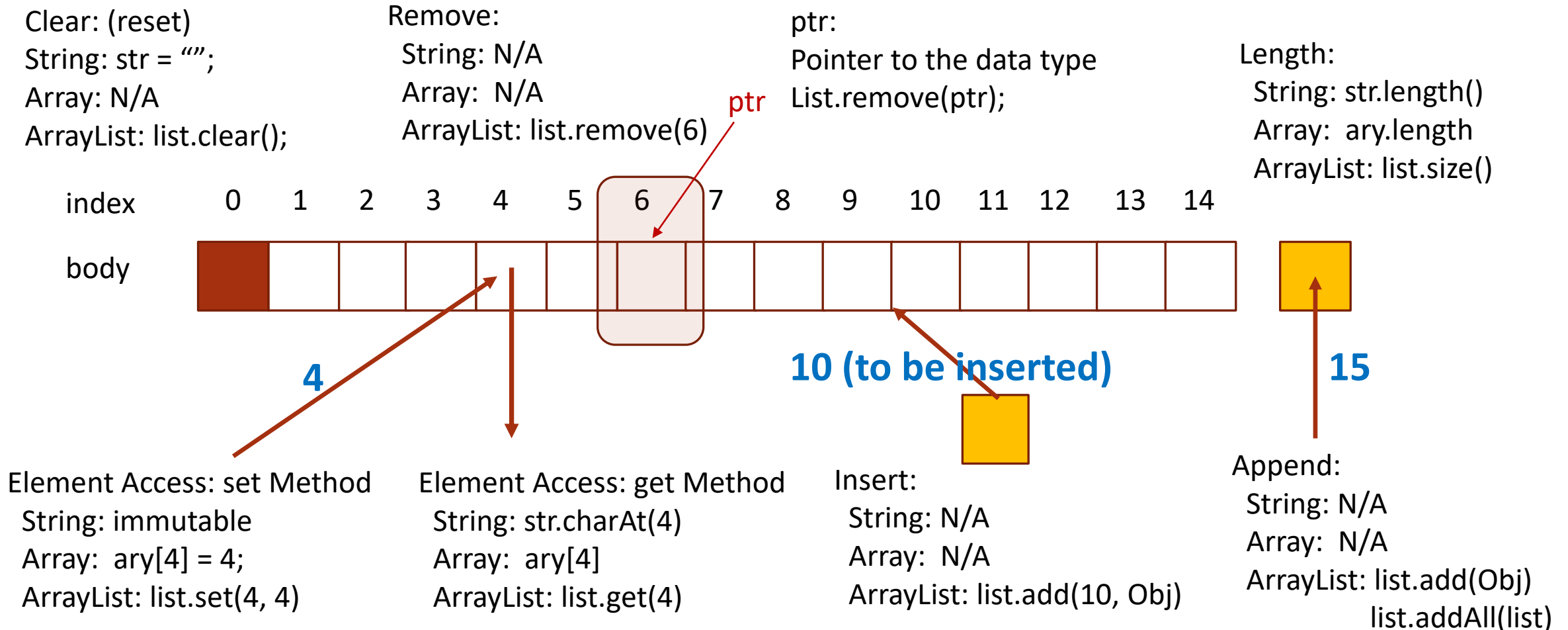
	Array of Character	String
Declaration	<code>char[] chary = {'A', 'B', 'C'};</code>	<code>String str = "ABC";</code>
New Object	<code>char[] chary = new char[3];</code>	<code>String str = new String("ABC");</code>
Access to Elements	<code>chary[2]</code>	<code>str.charAt(2)</code>
Change Content?	Yes	Immutable
Length	<code>chary.length</code>	<code>str.length()</code>
Partial elements	none	<code>substring(1,3)</code>
Easy Indexing	<code>chary[(a+b)/3*4-1+5/2]</code>	<code>str.charAt((a+b)/3*4-1+5/2)</code> can only fetch data
Object Traversal	Yes	No
Easy for <code>println()</code> ?	No	Yes
Concatenation?	No	Yes <code>System.out.println(str+str1+str2)</code>
Application	Tabularize data	Message Processing
Sorting of Elements	Yes	No
Adding new elements	No	No, but allow concatenation to create new string

# Differences and Similarities between Arrays and ArrayList



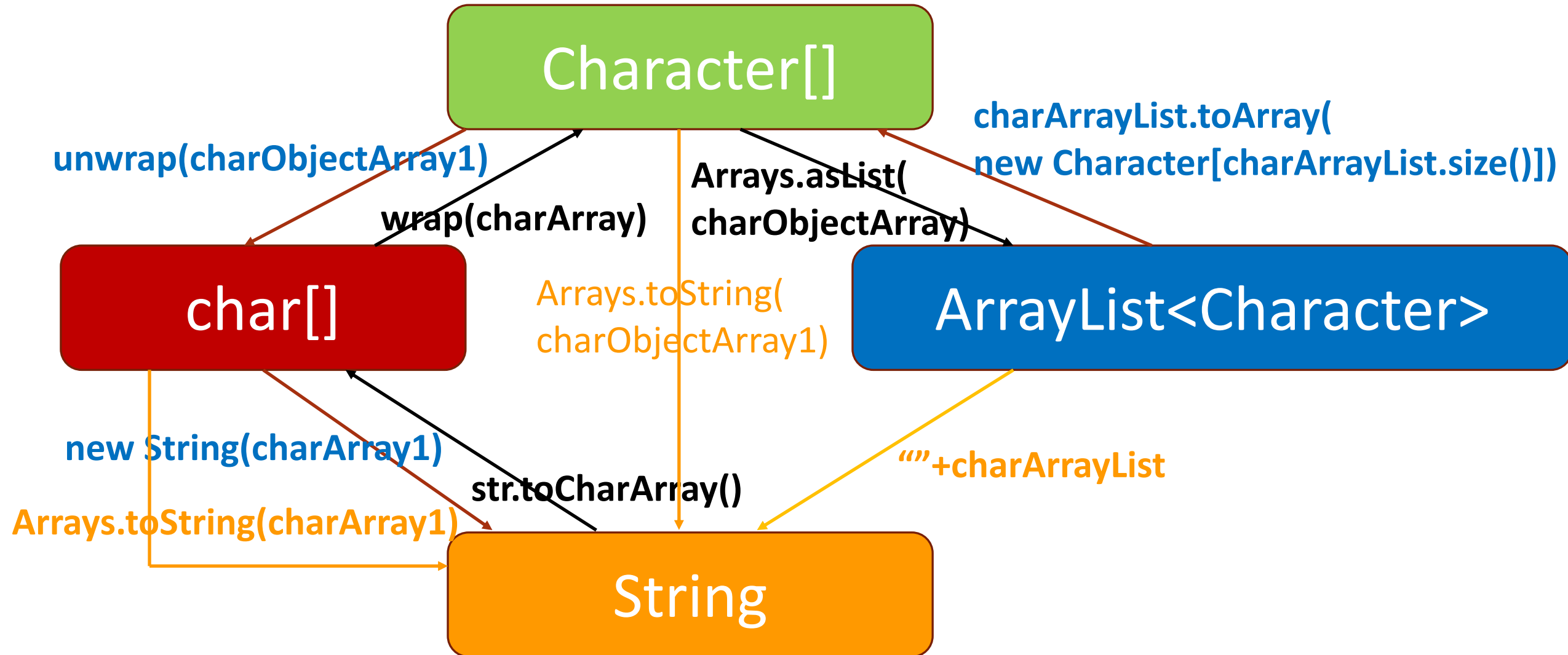
<i>Operation</i>	<i>Array</i>	<i>ArrayList</i>
Creating an array/ArrayList <code>ArrayList&lt;&gt;();</code>	<code>String[] a = new String[10]</code>	<code>ArrayList&lt;String&gt; list = new</code>
Accessing an element	<code>a[index]</code>	<code>list.get(index);</code>
Updating an element	<code>a[index] = "London";</code>	<code>list.set(index, "London");</code>
Returning size	<code>a.length</code>	<code>list.size();</code>
Adding a new element		<code>list.add("London");</code>
Inserting a new element		<code>list.add(index, "London");</code>
Removing an element		<code>list.remove(index);</code>
Removing an element		<code>list.remove(Object);</code>
Removing all elements		<code>list.clear();</code>

# Operation for String, Array and ArrayList





# Conversion Among String, char[], Character[] and ArrayList<Character>



## SECTION 6

# Interfaces for AP

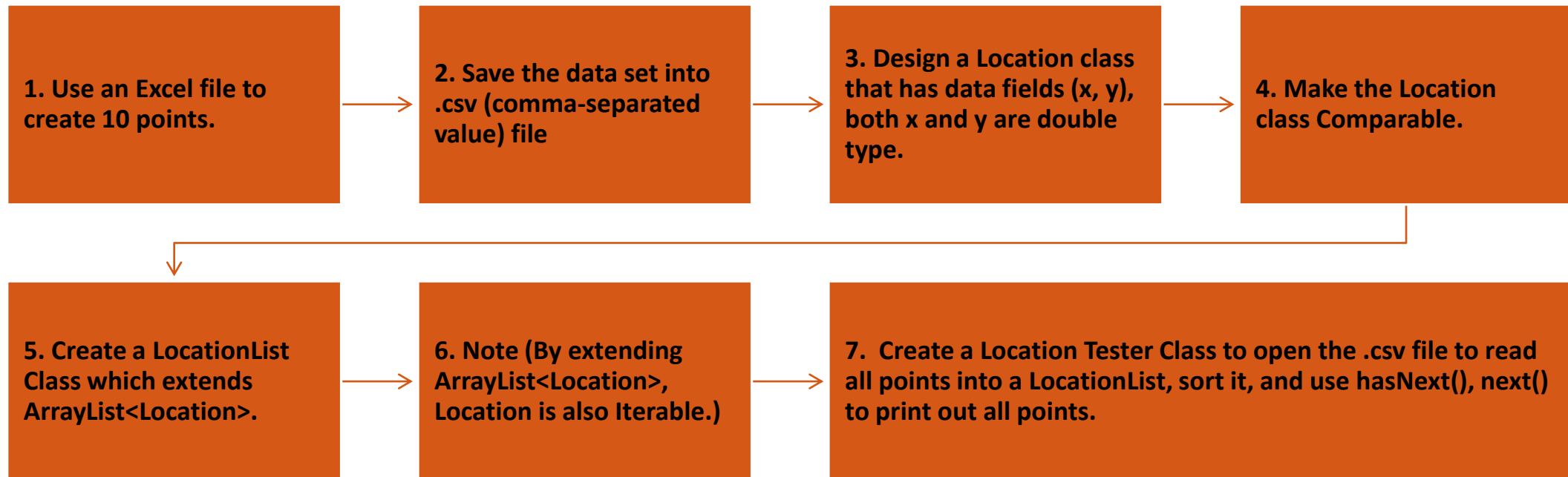


## Interfaces covered in AP

---

- `Comparable<T>`: `int compareTo(T other)`
- `Iterable<T>`: `hasNext()`, `next()`, `remove()`

# Put a list of (x, y) Coordinate in Order.



# Location/LocationList/Tester Classes

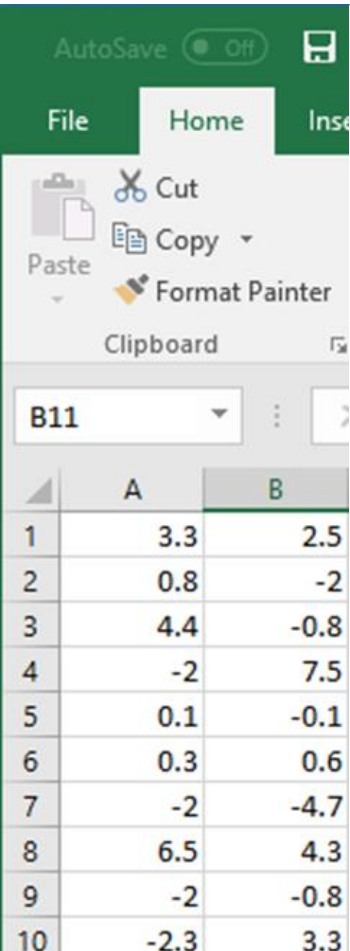
```
1 import java.util.ArrayList;
2 public class LocationList extends ArrayList<Location>{}
import java.util.*;
2 public class Location implements Comparable<Location>
3 {
4     double x, y;
5     Location(double x, double y){this.x = x; this.y=y; }
6     Location(){}
7     public double getDistance(){ return Math.sqrt(x*x + y*y); }
8     public int compareTo(Location that){
9         int r;
10        if (getDistance() > that.getDistance()) r = 1;
11        else if (getDistance() < that.getDistance()) r = -1;
12        else r = 0;
13        return r;
14    }
15    public String toString(){
16        String xstr = String.format("%5.2f",x);
17        String ystr = String.format("%5.2f",y);
18        return "("+xstr+", "+ystr+")";
19    }
20 }
```

```
1 import java.util.Scanner;
2 import java.io.File;
3 import java.util.Collections;
4
5 public class Tester
6 {
7     public static void main(String[] args) throws Exception {
8         System.out.print("\f");
9         Scanner in = new Scanner(new File("points.csv"));
10        LocationList list = new LocationList();
11        while (in.hasNext()){
12            String line = in.nextLine();
13            String[] tokens = line.trim().split(",");
14            double x = Double.parseDouble(tokens[0]);
15            double y = Double.parseDouble(tokens[1]);
16            Location z = new Location(x, y);
17            list.add(z);
18        }
19
20        Collections.sort(list);
21        System.out.println(list);
22    }
23 }
```

# The execution result:

**Output:** (This string output should be in one line. It has been modified for educational purpose.)

[( 0.10, -0.10), ( 0.30, 0.60), ( 0.80, -2.00), (-2.00, -0.80),  
(-2.30, 3.30), ( 3.30, 2.50), ( 4.40, -0.80), (-2.00, -4.70),  
(-2.00, 7.50), ( 6.50, 4.30)]



	A	B
1	3.3	2.5
2	0.8	-2
3	4.4	-0.8
4	-2	7.5
5	0.1	-0.1
6	0.3	0.6
7	-2	-4.7
8	6.5	4.3
9	-2	-0.8
10	-2.3	3.3

```

1 import java.util.Scanner;
2 import java.io.File;
3 import java.util.Collections;
4 import java.util.Iterator;
5 public class Tester2
6 {
7     public static void main(String[] args) throws Exception {
8         System.out.print("\f");
9         Scanner in = new Scanner(new File("points.csv"));
10        LocationList list = new LocationList();
11        while (in.hasNext()){
12            String line = in.nextLine();
13            String[] tokens = line.trim().split(",");
14            double x = Double.parseDouble(tokens[0]);
15            double y = Double.parseDouble(tokens[1]);
16            Location z = new Location(x, y);
17            list.add(z);
18        }
19
20        Collections.sort(list);
21        Iterator j = list.iterator();
22        int i=0;
23        System.out.print("[");
24        while (j.hasNext()){
25            if (i%3==0) { System.out.print(j.next());}
26            else if (i%3==2) { System.out.print(", "+j.next()+"\n");}
27            else { System.out.print(", "+j.next());}
28            i++;
29        }
30        System.out.println("]");
31    }
32 }

```

## Output:

```

[( 0.10, -0.10), ( 0.30, 0.60), ( 0.80, -2.00),
(-2.00, -0.80), (-2.30, 3.30), ( 3.30, 2.50),
( 4.40, -0.80), (-2.00, -4.70), (-2.00, 7.50),
( 6.50, 4.30)]

```

## This program gives example for the following things:

- Math: Math.sqrt()
- String: +, trim(), split()
- Object: toString()
- Double: Double.parseDouble()
- Comparable: compareTo()
- Collections: sort() (ArrayList)
- Iterable: hasNext(), next()
- Iterator: iterator()
- Inheritance by extends/implements