Mathematical Functions, Characters, and Strings

CSE 114: Introduction to Object-Oriented Programming

Paul Fodor

Stony Brook University

http://www.cs.stonybrook.edu/~cse114

Contents

- Static methods
- The Math Class API
  - Trigonometric methods
  - Exponent methods
  - Rounding methods
  - min, max, abs, and random methods
  - The random Method
- Characters
- The String Type
  - Reading a String from the Console
  - Useful String functions
    - length, charAt, concat, substring, equals, compareTo, indexOf
  - Strings are immutable!
  - Comparing Strings
  - Interned Strings
  - Conversion between Strings and Numbers
  - Formatting the Output

Static methods

- Remember the main method header?

```
public static void main(String[] args)
```

  - What does `static` **mean?**
    - associates the method with the class (NOT objects instances of that class)
    - any method can call a **static method either:**
      - directly from within same class OR
      - using the class name from outside class (if the method is visible, e.g., **`public`**)
  - The Application Programming Interface (API) is the list of all public members of a class

The `Math` Class API

- Class constants (always static):
  - `PI`
  - `E`
- Class static methods:
  - Trigonometric methods
  - Exponent methods
  - Rounding methods
  - min, max, abs, and random methods

```
public class Test {
  public static void main(String[] args) {
    System.out.println( Math.PI );
    System.out.println( Math.E );
  }
}
```

Output:

```
3.141592653589793
2.718281828459045
```

Trigonometric Methods

- sin(double a)
- cos(double a)
- tan(double a)
- acos(double a)
- asin(double a)
- atan(double a)

Radians

- **Examples:**

```
Math.sin(0) returns 0.0
Math.sin(Math.PI / 6)
   returns ~0.5
Math.sin(Math.PI / 2)
   returns 1.0
Math.cos(0) returns 1.0
Math.cos(Math.PI / 6)
   returns ~0.866
Math.cos(Math.PI / 2)
   returns ~0
```

```
public class Test {
  public static void main(String[] args) {
    System.out.println( Math.sin(0) );
    System.out.println( Math.sin(Math.PI / 6) );
    System.out.println( Math.sin(Math.PI / 2) );
    System.out.println( Math.cos(0) );
    System.out.println( Math.cos(Math.PI / 6) );
    System.out.println( Math.cos(Math.PI / 2) );
  }
}
```

```
0.0
0.49999999999999994
1.0
1.0
0.8660254037844387
6.123233995736766E-17
```

Exponent Methods

- **`exp(double a)`**

  Returns `e` raised to the power of `a`.

- **`log(double a)`**

  Returns the natural logarithm of `a`.

- **`log10(double a)`**

  Returns the 10-based logarithm of `a`.

- **`pow(double a, double b)`**

  Returns `a` raised to the power of `b`.

- **`sqrt(double a)`**

  Returns the square root of `a`.

- **Examples:**

```
Math.exp(1) returns 2.71
Math.log(2.71)
   returns 1.0
Math.pow(2, 3)
   returns 8.0
Math.pow(3, 2)
   returns 9.0
Math.pow(3.5, 2.5)
   returns 22.91765
Math.sqrt(4) returns 2.0
Math.sqrt(10.5)
   returns 3.24
```

```java
public class Test {
  public static void main(String[] args) {
    System.out.println( Math.exp(1) );
    System.out.println( Math.log(2.71) );
    System.out.println( Math.pow(2, 3) );
    System.out.println( Math.pow(3, 2) );
    System.out.println( Math.sqrt(4) );
    System.out.println( Math.sqrt(10.5) );
  }
}
```

```
2.718281828459045
0.9969486348916096
8.0
9.0
2.0
3.24037034920393
```

Rounding Methods

- `double ceil(double x)`

  x rounded up to its nearest integer. This integer is returned as a double value.

- `double floor(double x)`

  x is rounded down to its nearest integer. This integer is returned as a double value.

- `double rint(double x)`

  x is rounded to its nearest integer. If x is equally close to two integers, the even one is returned as a double.

- `int round(float x)`

  Return (int)Math.floor(x+0.5).

- `long round(double x)`

  Return (long)Math.floor(x+0.5).

Rounding Methods Examples

```
Math.ceil(2.1) returns 3.0
Math.ceil(2.0) returns 2.0
Math.ceil(-2.0) returns -2.0
Math.ceil(-2.1) returns -2.0
Math.floor(2.1) returns 2.0
Math.floor(2.0) returns 2.0
Math.floor(-2.0) returns -2.0
Math.floor(-2.1) returns -3.0
Math.round(2.6f) returns 3
Math.round(2.0) returns 2 (long)
Math.round(-2.0f) returns -2
Math.round(-2.6) returns -3 (long)
```

```
public class Test {
  public static void main(String[] args) {
    System.out.println( Math.ceil(2.1) );//3.0
    System.out.println( Math.ceil(2.0) );//2.0
    System.out.println( Math.ceil(-2.0) );//-2.0
    System.out.println( Math.ceil(-2.1) );//-2.0
    System.out.println( Math.floor(2.1) );//2.0
    System.out.println( Math.floor(2.0) );//2.0
    System.out.println( Math.round(2.6f) );//3
    System.out.println( Math.round(2.0) );//2
    System.out.println( Math.round(-2.0f) );//-2
    System.out.println( Math.round(-2.6) );//-3
  }
}
```

min, max, and abs

- **max(a, b)** and **min(a, b)**

  Returns the maximum or minimum of two parameters.

- **abs(a)**

  Returns the absolute value of the parameter.

- **random()**

  Returns a random `double` value
  in the range [0.0, 1.0).

- **Examples:**

```
Math.max(2, 3)
  returns 3
Math.max(2.5, 3)
  returns 3.0
Math.min(2.5,3.6)
  returns 2.5
Math.abs(-2)
  returns 2
Math.abs(-2.1)
  returns 2.1
```

```
public class Test {
  public static void main(String[] args) {
    System.out.println( Math.max(2, 3) );
    System.out.println( Math.max(2.5, 3) );
    System.out.println( Math.min(2.5, 3.6) );
    System.out.println( Math.min(2, 3) );
    System.out.println( Math.abs(-2) );
    System.out.println( Math.abs(-2.1) );
  }
}
```

```
3
3.0
2.5
2
2
2.1
```

The **random** Method

Generates a random **double** value greater than or equal to 0.0 and less than 1.0 (**0 <= Math.random() < 1.0**)

Examples:

`(int)(Math.random() * 10)` → `Returns a random integer between 0 and 9.`

`50 + (int)(Math.random() * 50)` → `Returns a random integer between 50 and 99.`

In general,

`a + Math.random() * b` → `Returns a random number between a and a + b, excluding a + b.`

```
public class Test {
  public static void main(String[] args) {
    System.out.println( Math.random() * 10 );//[0,10)
    System.out.println( (int)(Math.random() * 10) );
                        // {0,1,2,3,4,5,6,7,8,9}
    System.out.println( 50 + (int)(Math.random() * 50) );
                        // {50, 51, …, 99}
    int a = 50, b = 50;
    System.out.println( a + (int)(Math.random() * b) );
                        // {a, a+1, …, a+b-1}
                        // {50, 51, …, 99}
  }
}
```

```
2.5056436465195766
7
58
69
```

Generating Random Characters

```
(char)((int)'a' + Math.random() * ((int)'z' - (int)'a' + 1))
```

- However, all numeric operators can be applied to the char operands
- The char operand is also cast into a higher number type if the other operand is a number
- So, the preceding expression can be simplified as follows:

```
(char)('a' + Math.random() * ('z' - 'a' + 1))
```

```
public class Test {
  public static void main(String[] args) {
    System.out.println( (char)((int)'a' +
      Math.random() * ((int)'z' - (int)'a' + 1)) );
    System.out.println( (char)('a' +
      Math.random() * ('z' - 'a' + 1)) );
  }
}
```

```
d
v
```

ASCII Code for Commonly Used Characters

Characters	Code Value in Decimal	Unicode Value
'0' to '9'	48 to 57	\u0030 to \u0039
'A' to 'Z'	65 to 90	\u0041 to \u005A
'a' to 'z'	97 to 122	\u0061 to \u007A

There is no need to remember them since we can do all mathematical operations with characters:

```
(char)('a' + Math.random() * ('z' - 'a' + 1))
```

```
'0' <= c && c <= '9'
```

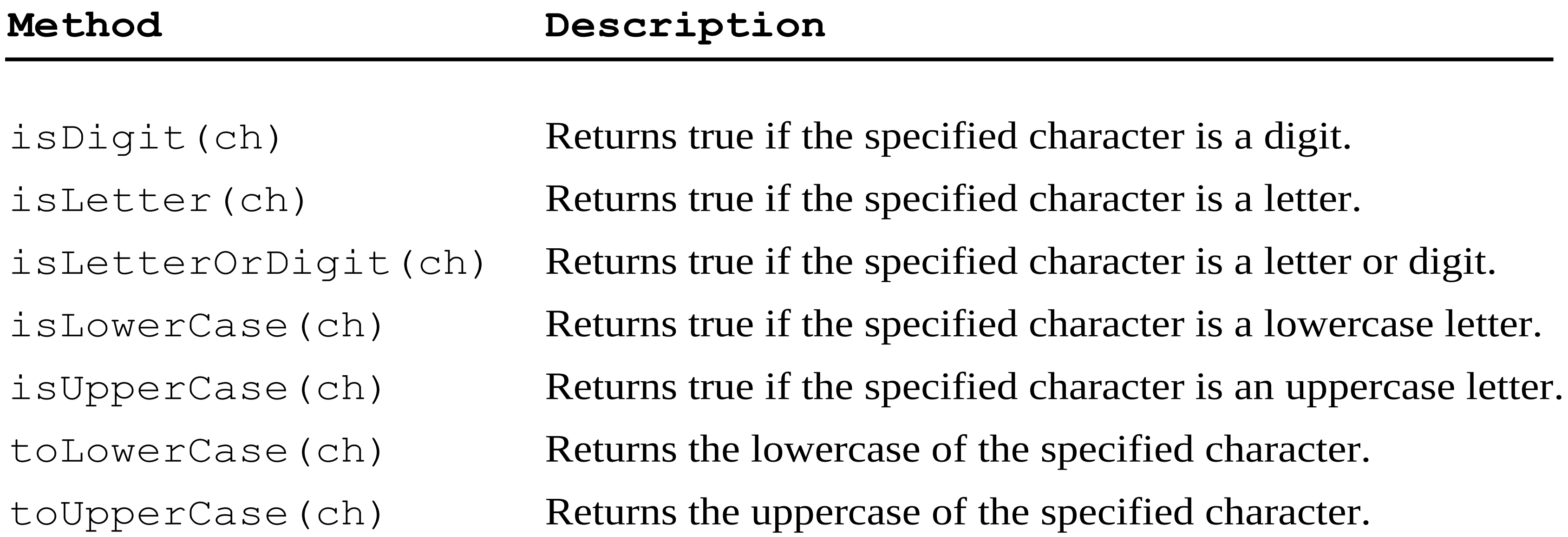
Comparing and Testing Characters

```
if ('A' <= ch && ch <= 'Z')
  System.out.println(ch + " is an uppercase letter");

if ('a' <= ch && ch <= 'z')
  System.out.println(ch + " is a lowercase letter");

if ('0' <= ch && ch <= '9')
  System.out.println(ch + " is a numeric character");
```

Methods in the Character Class



Method	Description
`isDigit(ch)`	Returns true if the specified character is a digit.
`isLetter(ch)`	Returns true if the specified character is a letter.
`isLetterOrDigit(ch)`	Returns true if the specified character is a letter or digit.
`isLowerCase(ch)`	Returns true if the specified character is a lowercase letter.
`isUpperCase(ch)`	Returns true if the specified character is an uppercase letter.
`toLowerCase(ch)`	Returns the lowercase of the specified character.
`toUpperCase(ch)`	Returns the uppercase of the specified character.

Comparing and Testing Characters

```
if (Character.isUpperCase(ch))
  System.out.println(ch + " is an uppercase letter");

if (Character.isLowerCase(ch))
  System.out.println(ch + " is a lowercase letter");

if (Character.isDigit(ch))
  System.out.println(ch + " is a numeric character");
```

The String Type

- The **`char`** type only represents one character:

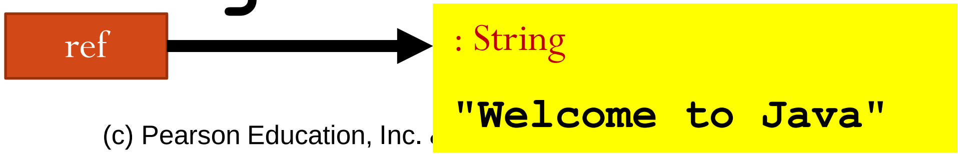
```
char ch = 'a';
```

- To represent a string of characters, use the data type called String.

String is a predefined class in the Java library just like the System class http://java.sun.com/javase/8/docs/api/java/lang/String.html

- The String type is NOT a primitive type.
  - The String type is a *reference type*.
    - A String variable is a *reference variable*, an *address* (also called *pointer*) which points to an object storing the value or actual text

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



Reading a String from the Console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter three words separated by spaces:");
    // one two three
String s1 = input.next(); // "one"
String s2 = input.next(); // "two"
String s3 = input.next(); // "three"
System.out.println("s1 is " + s1);
System.out.println("s2 is " + s2);
System.out.println("s3 is " + s3);
```

Reading a String from the Console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a line:");
    // one two three
String s = input.nextLine();
    // "one two three"
System.out.println("s is " + s);
    // s is one two three
```

Reading a single Character from the Console

```
Scanner input = new Scanner(System.in);
System.out.print("Enter a character: ");

String s = input.nextLine();
char ch = s.charAt(0);

System.out.print("The character entered is "+ch);
```

Useful String functions

- **length, charAt, concat, substring, equals, equalsIgnoreCase, compareTo, compareToIgnoreCase, startsWith, endsWith, indexOf, lastIndexOf.**

Finding a String Length

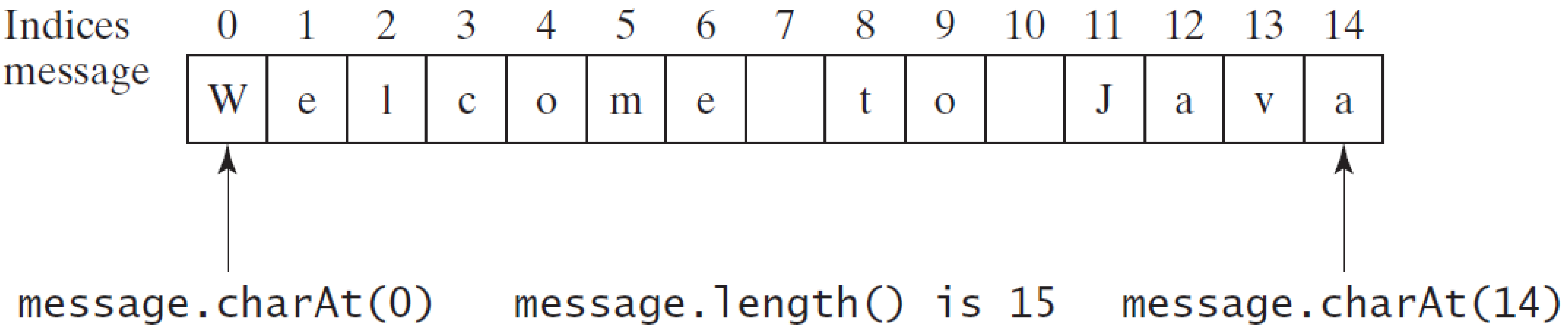
- Finding string length using the **`length()`** method:

```
String message = "Welcome to Java";
System.out.print( message.length());
   // prints 15
```

Getting Characters from a String

- Each character is stored at an index:

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



Indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
message	W	e	l	c	o	m	e		t	o		J	a	v	a

message.charAt(0)   message.length() is 15   message.charAt(14)

```
System.out.println(
   "The first character in message is "
   + message.charAt(0));
```

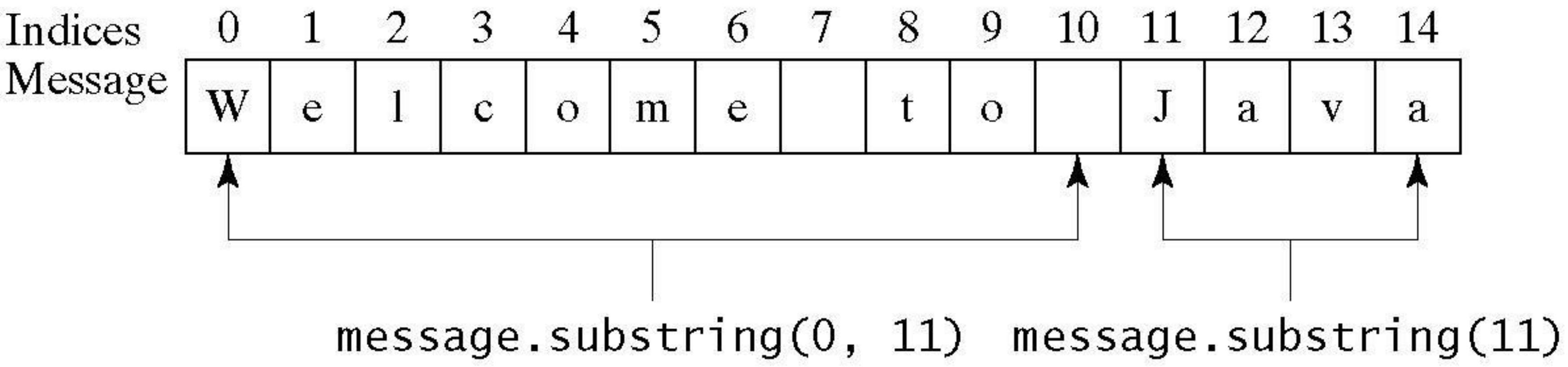
String Concatenation

- "+" is used for making a new string by concatenating strings:

```
// Three strings are concatenated
String message = "Welcome " + "to " + "Java";
// String Chapter is concatenated with number 2
String s = "Chapter" + 2; // s becomes Chapter2
// String Supplement is concatenated with character B
String s1 = "Supplement" + 'B';
     // s1 becomes SupplementB
String s2 = 1 + 2 + "ABC";
     // s2 become "3ABC"
String s2 = "" + 1 + 2 + "ABC";
     // s2 become "12ABC"
```

Obtaining Substrings

Method	Description
`substring(beginIndex)`	Returns this string's substring that begins with the character at the specified `beginIndex` and extends to the end of the string, as shown in Figure 4.2.
`substring(beginIndex, endIndex)`	Returns this string's substring that begins at the specified `beginIndex` and extends to the character at index `endIndex – 1`, as shown in Figure 9.6. Note that the character at `endIndex` is not part of the substring.



Indices	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Message	W	e	l	c	o	m	e		t	o		J	a	v	a

`message.substring(0, 11)`

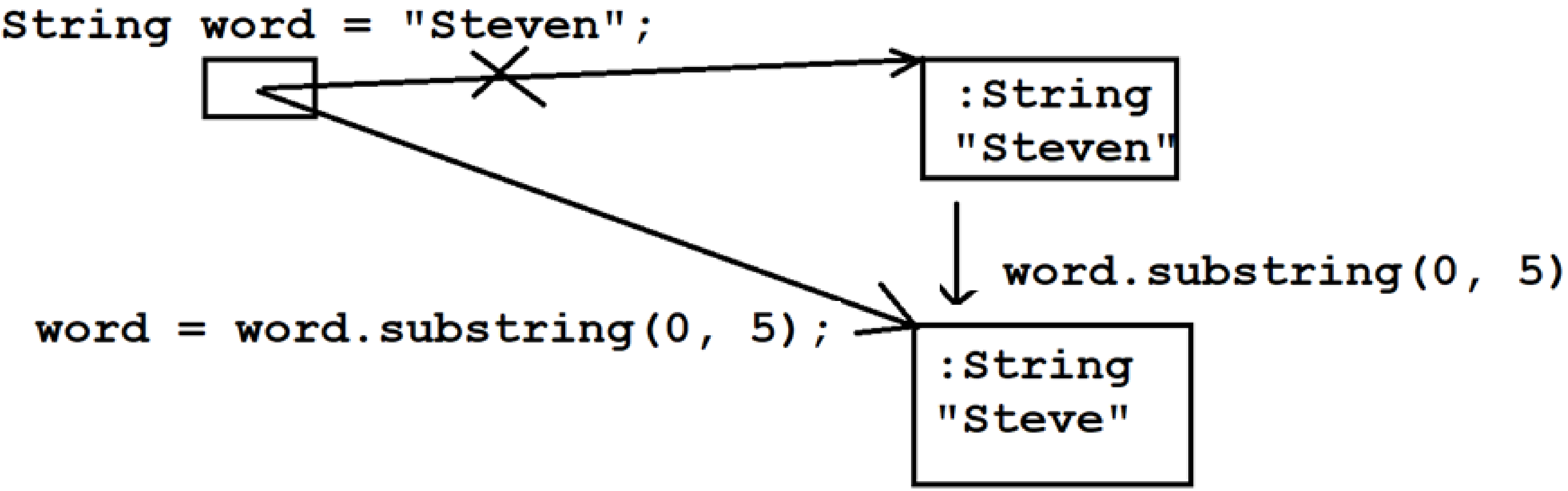
`message.substring(11)`

Strings are immutable!

- There are no methods to change them once they have been created
  - any new assignment will assign a new String reference to the old variable

```
String word = "Steven";
word = word.substring(0, 5);
```

  - the variable word is now a reference to a new String that contains "Steve"



```
String word = "Steven";
```

:String
"Steven"

word.substring(0, 5)

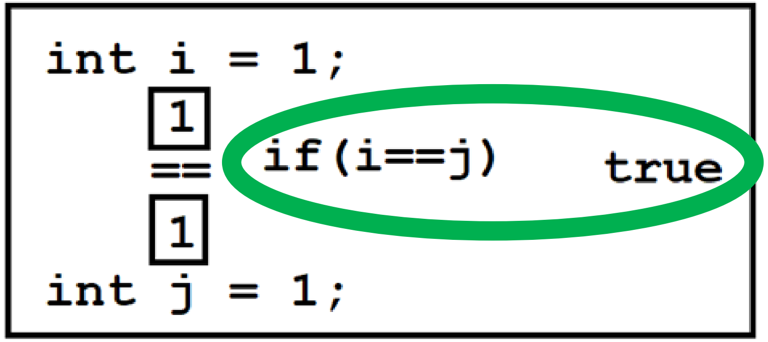
```
word = word.substring(0, 5);
```

:String
"Steve"

Comparing Strings

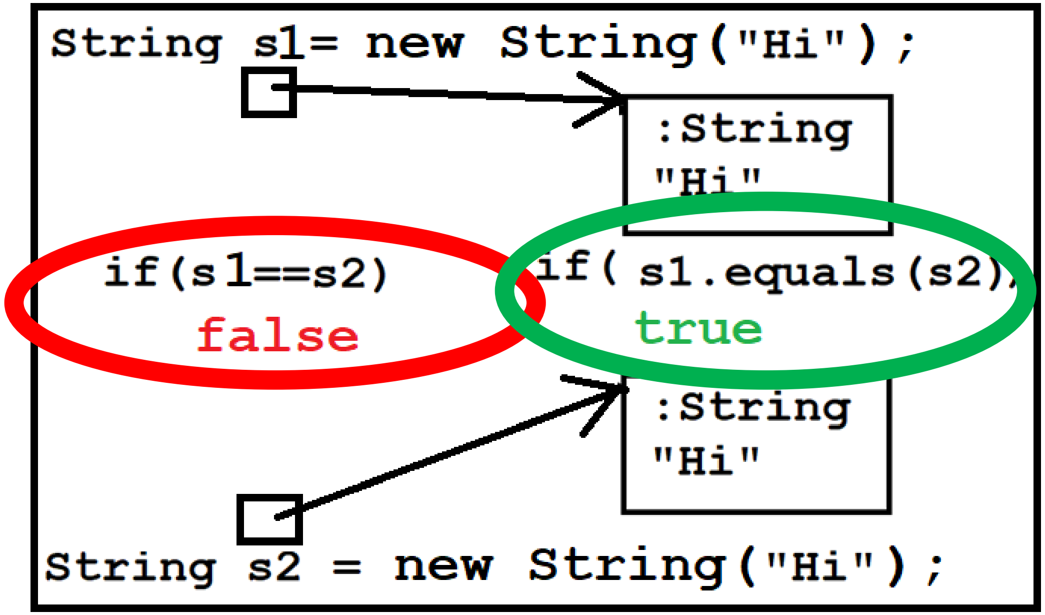
- Don’t use ‘==’ to compare Strings
  - it compares their memory addresses and not actual strings (character sequences)
- Instead use the **`equals`** method supplied by the String class:
  - **`s.equals(t)`**
    - returns **`true`** if **`s`** and **`t`** have same letters and sequence
    - **`false`** otherwise

== for Primitive vs. equals for Reference Types



```
int i = 1;
```

```
int j = 1;
```



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

```
String s2 = new String("Hi");
```

Comparing Strings

```
String word1 = new String("Hello");
String word2 = new String("Hello");
if (word1 == word2){
  System.out.println(true);
} else {
  System.out.println(false);
}
```

```
false
```

Two different references/addresses

Comparing Strings

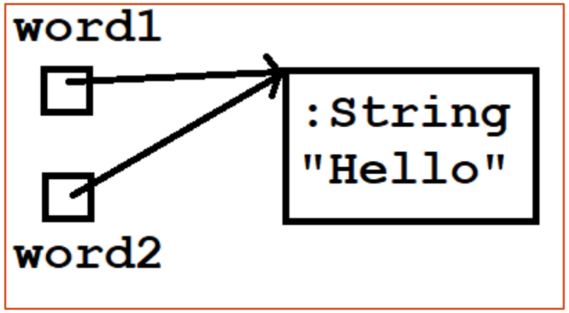
```
String word1 = new String("Hello");
String word2 = new String("Hello");
if (word1.equals(word2)){
  System.out.println(true);
} else {
  System.out.println(false);
}
```

true

compares the contents: "Hello" with "Hello"

Interned Strings

String interning is a method of storing only one copy of each distinct string value, i.e., the distinct values are stored in a pool of unique strings - All compile-time constant strings in Java are automatically interned using this method.



```
String word1 = "Hello";
String word2 = "Hello";
if (word1 == word2){
  System.out.println(true);
} else {
  System.out.println(false);
}
```

true

- Interned Strings: only one instance of "Hello" is stored
  - so word1 and word2 will have the same address

Interned Strings

- **equals** still works as it is supposed to work:

```
String word1 = "Hello";
String word2 = "Hello";
if (word1.equals(word2)){
  System.out.println(true);
} else {
  System.out.println(false);
}
```

Also **true**

So, we always use equals.

Interned Strings

```
String word1 = new String("Hello");
String word2 = "Hello";
if (word1.equals(word2)){
  System.out.println(true);
} else {
  System.out.println(false);
}
```

```
                    true
```

Interned Strings

```
String word1 = "Hello";
String word2 = new String("Hello");
if (word1.equals(word2)){
 System.out.println(true);
} else {
 System.out.println(false);
}
```

```
true
```

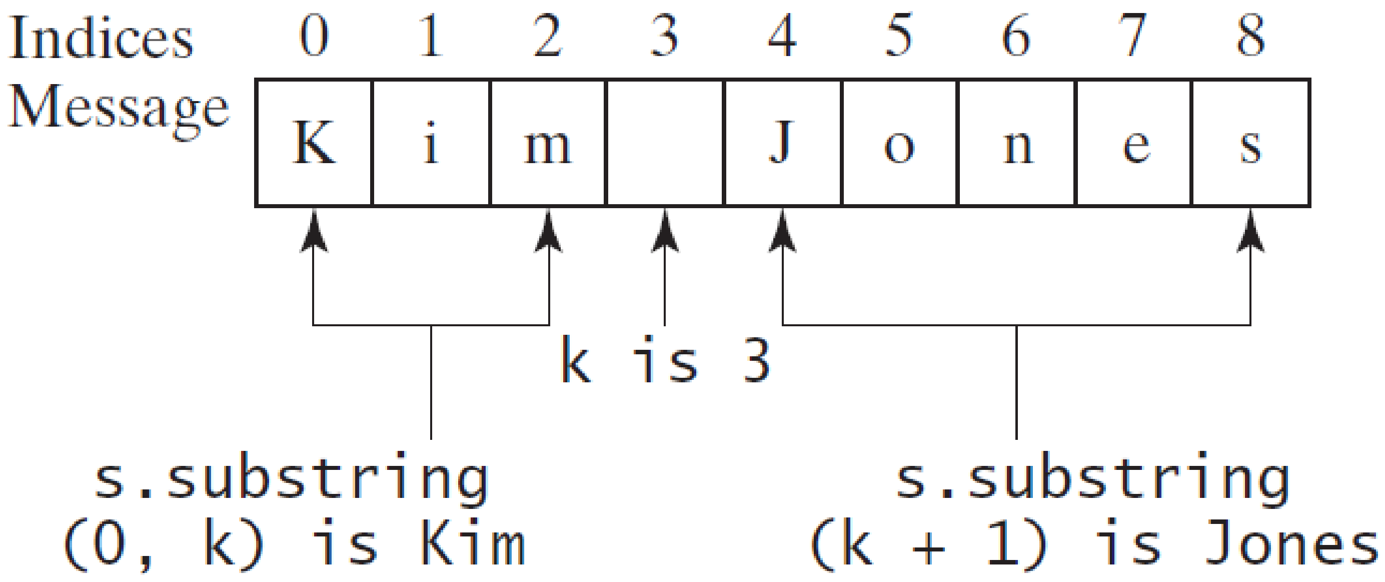
Comparing Strings

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

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` after `fromIndex` in the string. Returns `-1` if not matched.
`indexOf(s)`	Returns the index of 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 after `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` before `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` before `fromIndex`. Returns `-1` if not matched.

Finding a Character or a Substring in a String



Indices	0	1	2	3	4	5	6	7	8
Message	K	i	m		J	o	n	e	s

k is 3

s.substring
(0, k) is Kim

s.substring
(k + 1) is Jones

```
int k = s.indexOf(' ');  //3
String firstName = s.substring(0, k);
String lastName = s.substring(k + 1);
```

Conversion between Strings and Numbers

```
String intString = "15";
String doubleString = "56.77653";

int intValue =
     Integer.parseInt(intString);
double doubleValue =
     Double.parseDouble(doubleString);

String s1 = "" + intValue;
String s2 = "" + doubleValue;
```

Formatting the Output

The printf statement:

```
System.out.printf(format, items);
```

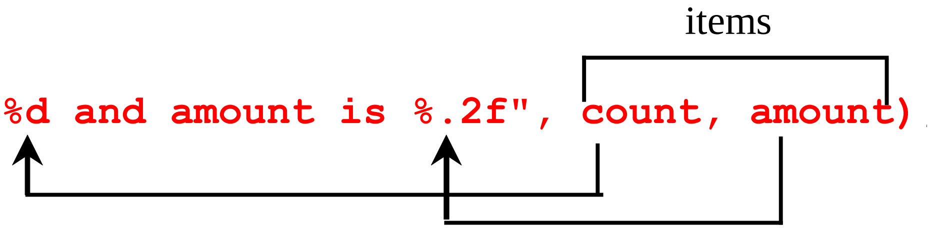
format is a string that may consist of substrings and format **specifiers:**

- A format specifier begins with a percent sign (%) and specifies how an item should be displayed: a numeric value, character, boolean value, or a string

Frequently-Used Specifiers

Specifier	Output	Example
<u>%b</u>	a boolean value	true or false
<u>%c</u>	a character	'a'
<u>%d</u>	a decimal integer	200
<u>%f</u>	a floating-point number	45.460000
<u>%e</u>	a number in standard scientific notation	4.556000e+01
<u>%s</u>	a string	"Java is cool"

```
int count = 5;
double amount = 45.561899;
System.out.printf("count is %d and amount is %.2f", count, amount)
```



Displays: `count is 5 and amount is 45.56`