

Outline

Using Classes and Objects

Creating Objects

The String Class

The Character Class

The Random Class

The Math Classes

Formatting Output

Creating Objects

- A variable holds either a **primitive value** or a *reference* to an object
- A class name can be used as a type to declare an *object reference variable*

Creating Objects

- Consider the following two declarations:

```
int num;
```



```
// integer variable exists  
// but it is not initialized
```

```
String name;
```



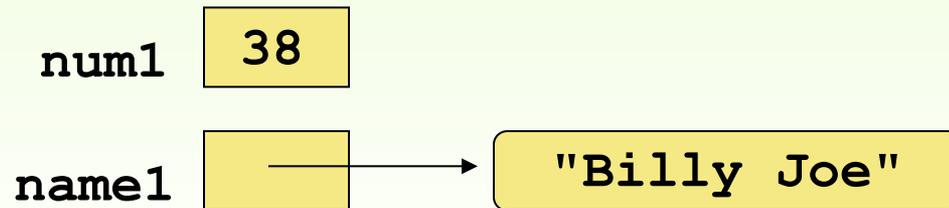
```
// No String object actually  
// exists yet!
```

- The first declaration creates a variable that holds an integer value.
- The second declaration creates a String variable that holds a *reference* to a String object.

References

- A primitive variable contains **the value** itself, but an object variable contains **the address of the object**
- An object reference can be thought of as a pointer to the location of the object
- Consider the following two declarations:

```
int num1 = 38;  
String name1 = new String("Billy Joe");
```



Memory for a String object

- Suppose that name1 contains the memory address AAA1000 in hexadecimal representation. Then, the following memory allocation is true:

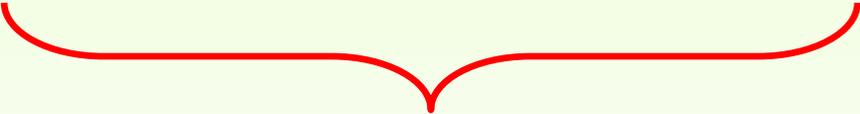
Each character in Java is stored in 2 bytes

AAA1000	'B'
AAA1002	'i'
AAA1004	'l'
AAA1006	'l'
AAA1008	'y'
AAA100A	' '
AAA100C	'J'
AAA100E	'o'
AAA1010	'e'

Creating Objects

- We use the **new** operator to create an object
- Creating an object is called *instantiation*
- An object is an *instance* of a particular class

```
name1 = new String ("Billy Joe");
```



This calls the String *constructor*, which is a special method that sets up the object

Assignment Revisited

- The act of assignment takes a copy of a value and stores it in a variable
- For primitive types:

Before:

num1	38
num2	96

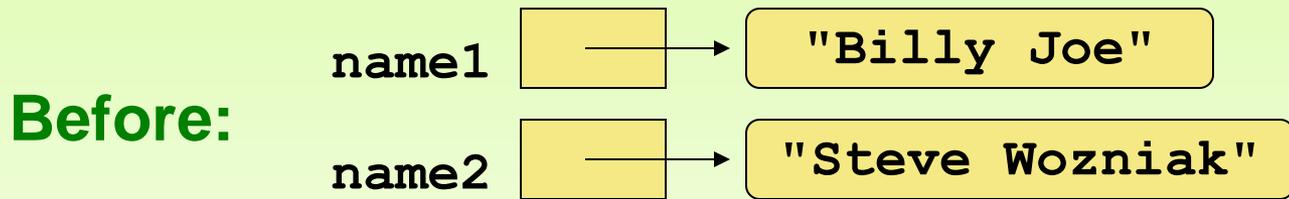
```
num2 = num1;
```

After:

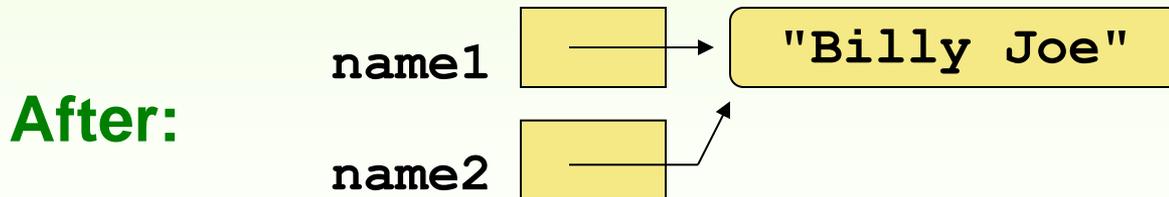
num1	38
num2	38

Reference Assignment

- For object references, assignment copies the address:



```
name2 = name1;
```



Aliases

- Two or more references that refer to the same object are called *aliases* of each other
- That creates an interesting situation: one object can be accessed using multiple reference variables
- Aliases can be useful, but should be managed carefully
- Changing an object through one reference changes it for all of its aliases, because there is really only one object

Outline

Using Classes and Objects

Creating Objects



The String Class

The Character Class

The Random Class

The Math Classes

Formatting Output

Class Libraries

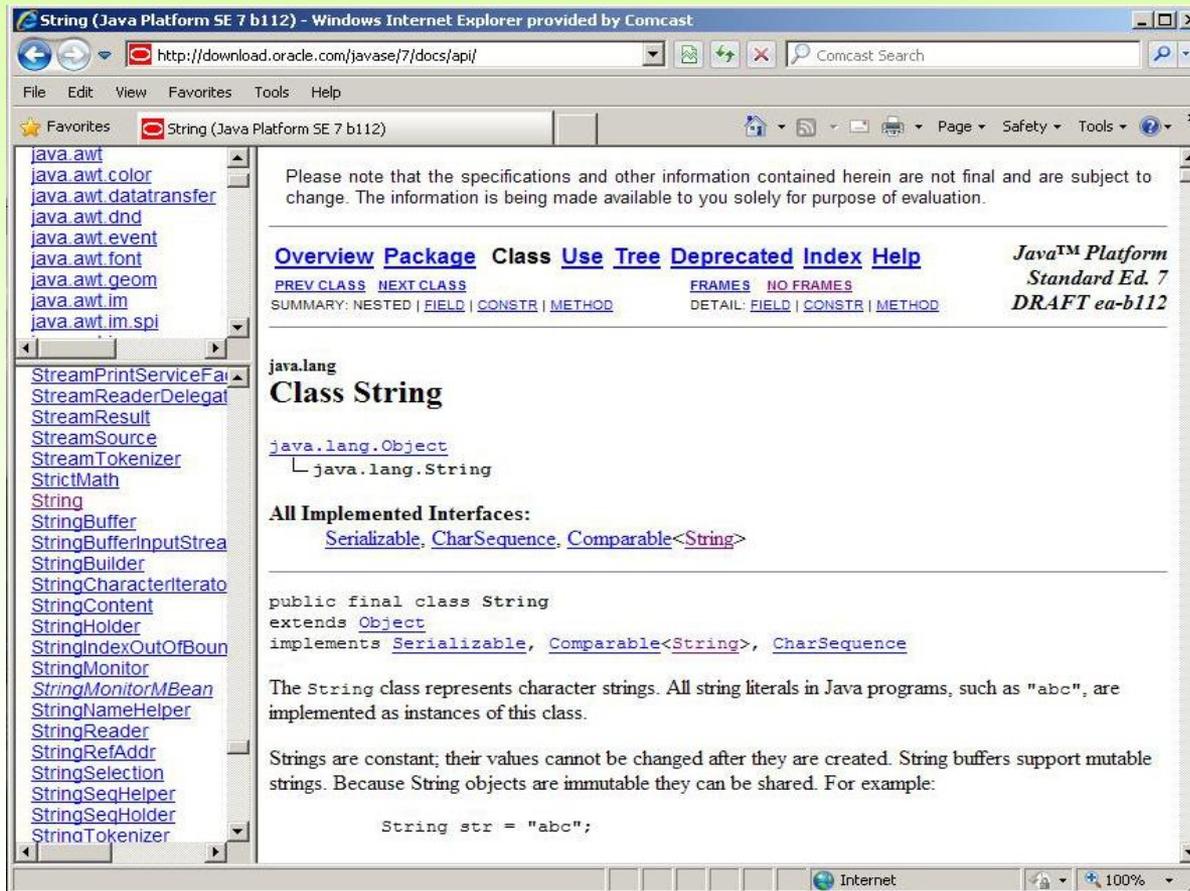
- A *class library* is a collection of classes that we can use when developing programs
- The *Java standard class library* is part of any Java development environment
- Various classes we've already used (`System`, `String`) are part of the Java standard class library

The Java API

- The Java class library is sometimes referred to as the Java API
- API stands for Application Programming Interface

The Java API

- Get comfortable navigating the online Java API documentation



The Import Declaration

- When you want to use a class from a package, you could use its *fully qualified name*

```
java.util.Scanner
```

- Or you can *import* the class, and then use just the class name

```
import java.util.Scanner;
```

- To import all classes in a particular package, you can use the * wildcard character

```
import java.util.*;
```

The String Class

- Because strings are so commonly used, **Java treats a string literal as a `String` object.** Thus the following statement is valid:

```
String title = "Billy Joe";
```

instead of

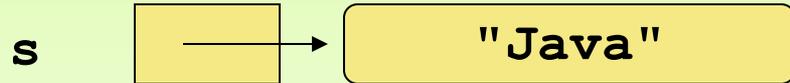
```
String title = new String ("Billy Joe");
```

- This is a **special syntax** that works only for strings
- Each string literal (enclosed in double quotes) represents a **`String`** object

String Objects

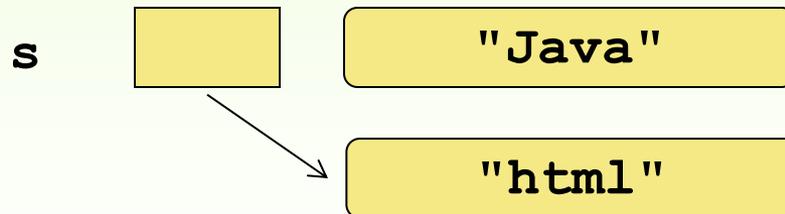
- A `String` object is *immutable*; its contents cannot be changed. Does the following code change the contents of the string?

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

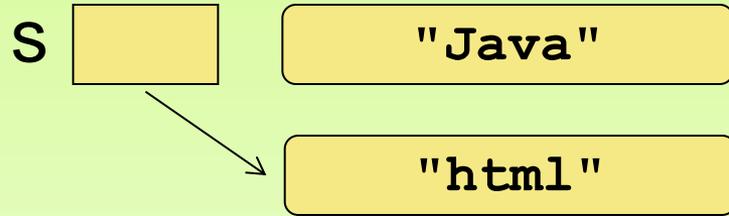


```
s = "html";
```

The answer is **NO**. Variable `s` now points to the new string object as shown:



String Objects



The unreferenced String object becomes a garbage. Luckily Java collects its own garbage. The 8 bytes occupied by "Java" is brought back to free memory space.

Some Useful Methods in String Class

String (String str)	Constructor: creates a new string object with the same characters as str.
int length ()	Returns the number of characters in this string.
char charAt (int index)	Returns the character at the specified index. (Index starts with 0.)
int compareTo (String str)	Returns an integer indicating if this string is lexically before (a negative return value), equal to (a zero return value), or lexically after (a positive return value) the string str.
String concat (String str)	Returns a new string consisting of this string concatenated with str.
boolean equals (String str)	Returns true if this string contains the same characters as str (including case) and false otherwise.
boolean equalsIgnoreCase (String str)	Returns true if this string contains the same characters as str (without regard to case) and false otherwise.
String replace (char oldChar, char newChar)	Returns a new string that is identical with this string except that every occurrence of oldChar is replaced by newChar.
String substring (int offset, int endIndex)	Returns a new string that is a subset of this string starting at index offset and extending through endIndex-1.
String toLowerCase()	Returns a new string identical to this string except all uppercase letters are converted to their lowercase equivalent.
String toUpperCase()	Returns a new string identical to this string except all lowercase letters are converted to their uppercase equivalent.
int indexOf (char ch)	Returns the index of the first occurrence of ch in this string. Returns -1 if ch is not found in this string.
int indexOf (String str)	Returns the index of the first occurrence of str in this string. Returns -1 if str is not found in this string.

Other String Class Methods in Java

int `lastIndexOf(char ch)` *or* `lastIndexOf(String str)`
Returns the index of the last match of the argument, or -1 if none exists.

boolean `startsWith(String str)`
Returns `true` if this string starts with `str`.

boolean `endsWith(String str)`
Returns `true` if this string starts with `str`.

String `trim()`
Returns a copy of this string with leading and trailing whitespace

String `toLowerCase()`
Returns a copy of this string with all uppercase characters changed to lowercase.

String `toUpperCase()`
Returns a copy of this string with all lowercase characters changed to uppercase

Checking Strings for Equality

- Many applications will require you to **test whether two strings are equal**, in the sense that they contain the same characters.
- Although it seems natural to do so, **you cannot use the == operator** for this purpose. While it is legal to write

```
if (s1 == s2) . . .
```



the if test will not have the desired effect. When you use == on two objects, it checks whether the objects are *identical*, which means that the references point to the same address.

- What you need to do instead is call the **equals** method:

```
if (s1.equals(s2)) . . .
```

String Class equals

Checking equality of two Strings

```
import java.util.*;
public class Program {
    public static void main(String[] args) {

        String physicist1 = "Albert Einstein";
        String physicist2 = "Max Planck";
        String physicist3 = "Albert Einstein";

        // Are any of the above Strings equal to one another?
        boolean equals1 = physicist1.equals(physicist2);
        boolean equals2 = physicist1.equals(physicist3);

        // Display the results of the equality checks.
        System.out.println("\"" + physicist1 + "\" equals \"" + physicist2 + "\"? "
            + equals1);
        System.out.println("\"" + physicist1 + "\" equals \"" + physicist3 + "\"? "
            + equals2);

        // Compare == with equals method
        Scanner scan = new Scanner (System.in);
        String physicist4 = scan.nextLine();
        System.out.println("\"" + physicist1 + "\" == \"" + physicist3 + "\"? " +
            (physicist1 == physicist3));
        System.out.println("\"" + physicist1 + "\" == \"" + physicist4 + "\"? " + (
            physicist1 == physicist4));

    }
}
```

String Class

Checking equality

```
import java.util.*;
public class Program {
    public static void main(String[] args) {

        String physicist1 = "Albert Einstein";
        String physicist2 = "Max Planck";
        String physicist3 = "Albert Einstein";

        // Are any of the above Strings equal to one another?
        boolean equals1 = physicist1.equals(physicist2);
        boolean equals2 = physicist1.equals(physicist3);

        // Display the results of the equality checks.
        System.out.println("\"" + physicist1 + "\" equals \"" + physicist2 + "\"? "
            + equals1);
        System.out.println("\"" + physicist1 + "\" equals \"" + physicist3 + "\"? "
            + equals2);

        // Compare == with equals method
        Scanner scan = new Scanner (System.in);
        String physicist4 = scan.nextLine();
        System.out.println("\"" + physicist1 + "\" == \"" + physicist3 + "\"? " +
            (physicist1 == physicist3));
        System.out.println("\"" + physicist1 + "\" == \"" + physicist4 + "\"? " + (
            physicist1 == physicist4));

    }
}
```

Sample Run

```
"Albert Einstein" equals "Max Planck"? false
"Albert Einstein" equals "Albert Einstein"? true
Albert Einstein
"Albert Einstein" == "Albert Einstein"? true
"Albert Einstein" == "Albert Einstein"? false
```

String Class equalsIgnoreCase

```
public class Program2 {
    public static void main(String[] args) {

        String physicist1 = "Albert Einstein";
        String physicist2 = "Max Planck";
        String physicist3 = "albert einstein";

        // Are any of the above Strings equal to one another?
        boolean equals1 = physicist1.equalsIgnoreCase(physicist2);
        boolean equals2 = physicist1.equalsIgnoreCase(physicist3);

        // Display the results of the equality checks
        System.out.println("\n" + physicist1 + "\" equals \"" +
            physicist2 + "\"? " + equals1);

        System.out.println("\n" + physicist1 + "\" equals \"" +
            physicist3 + "\"? " + equals2);

    }
}
```

String Class equalsIgnoreCase

The char at the given

Sample Run

```
"Albert Einstein" equals "Max Planck"? false  
"Albert Einstein" equals "albert einstein"? true
```

```
public class Program2 {  
    public static void main(String[] args) {  
  
        String physicist1 = "Albert Einstein";  
        String physicist2 = "Max Planck";  
        String physicist3 = "albert einstein";  
  
        // Are any of the above Strings equal to one another?  
        boolean equals1 = physicist1.equalsIgnoreCase(physicist2);  
        boolean equals2 = physicist1.equalsIgnoreCase(physicist3);  
  
        // Display the results of the equality checks  
        System.out.println("\"" + physicist1 + "\" equals \"" +  
            physicist2 + "\"? " + equals1);  
  
        System.out.println("\"" + physicist1 + "\" equals \"" +  
            physicist3 + "\"? " + equals2);  
  
    }  
}
```

Comparing Two Strings

- The `compareTo` method can also be used to compare two strings:

```
s1.compareTo(s2)
```

returns:

0 if s1 is equal to s2

<0 if s1 is lexicographically less than s2

>0 if s1 is lexicographically greater than s2

Converting, Replacing

`"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"`

String Indexes

- It is helpful to refer to a particular character within a string
- This can be done by specifying the character's numeric *index*
- The indexes begin at zero in each string
- In the string "Hello", the character 'H' is at index 0 and the 'o' is at index 4

String Class charAt

The **char** at the given index within the String.

```
public class Program3
{
    public static void main(String[] args)
    {
        String str = "Hello, World!";

        // Get the character at positions 0 and 12.
        char ch1 = str.charAt(0);
        char ch2 = str.charAt(12);

        // Print out the results
        System.out.println("The character at position 0 is " + ch1);
        System.out.println("The character at position 12 is " + ch2);

    }
}
```

String Class charAt

The **char** at the given index within the String.

```
public class Program3
{
    public static void main(String[] args)
    {
        String str = "Hello, World!";

        // Get the character at positions 0 and 12.
        char ch1 = str.charAt(0);
        char ch2 = str.charAt(12);

        // Print out the results
        System.out.println("The character at position 0 is " + ch1);
        System.out.println("The character at position 12 is " + ch2);

    }
}
```

Sample Run

The character at position 0 is H
The character at position 12 is !

How to get a char from the user using Scanner class?

As mentioned earlier, the Scanner class does not have a method that returns a char alone. Use the following to extract the first character from a String of 1 (or more..) characters:

```
char ch = scan.next().charAt(0);
```

- See [StringMutation.java](#)

```

//*****
//  StringMutation.java          Author: Lewis/Loftus
//
//  Demonstrates the use of the String class and its methods.
//*****

public class StringMutation
{
    //-----
    //  Prints a string and various mutations of it.
    //-----
    public static void main (String[] args)
    {
        String phrase = "Change is inevitable";
        String mutation1, mutation2, mutation3, mutation4;

        System.out.println ("Original string: \" " + phrase + "\"");
        System.out.println ("Length of string: " + phrase.length());

        mutation1 = phrase.concat (" , except from vending machines.");
        mutation2 = mutation1.toUpperCase();
        mutation3 = mutation2.replace ('E', 'X');
        mutation4 = mutation3.substring (3, 30);
    }
}

```

continued

continued

```
// Print each mutated string
System.out.println ("Mutation #1: " + mutation1);
System.out.println ("Mutation #2: " + mutation2);
System.out.println ("Mutation #3: " + mutation3);
System.out.println ("Mutation #4: " + mutation4);

System.out.println ("Mutated length: " + mutation4.length());
}
}
```

Output

Original string: "Change is inevitable"

Length of string: 20

Mutation #1: Change is inevitable, except from vending machines.

Mutation #2: CHANGE IS INEVITABLE, EXCEPT FROM VENDING MACHINES.

Mutation #3: CHANGX IS INXVITABLX, XXCXPT FROM VXNDING MACHINXS.

Mutation #4: NGX IS INXVITABLX, XXCXPT F

Mutated length: 27

```
        System.out.println ("Mutated length: " + mutation4.length());  
    }  
}
```

Quick Check

What output is produced by the following?

```
String str = "Space, the final frontier.";
System.out.println (str.length());
System.out.println (str.substring(7));
System.out.println (str.toUpperCase());
System.out.println (str.length());
```

Quick Check

What output is produced by the following?

```
String str = "Space, the final frontier.";
System.out.println (str.length());
System.out.println (str.substring(7));
System.out.println (str.toUpperCase());
System.out.println (str.length());
```

26

the final frontier.

SPACE, THE FINAL FRONTIER.

26

Extracting Substring

- The `substring` method makes it possible to extract a piece of a larger string by providing index numbers that determine the extent of the substring.
- The general form of the `substring` call is

```
str.substring(p1, p2);
```

where `p1` is the first index position in the desired substring and `p2` is the index position immediately following the last position in the substring.

- As an example, if you wanted to select the substring "ell" from a string variable `str` containing "hello, world" you would make the following call:

```
str.substring(1, 4);
```

String Example

- Reverse a string
 - Take input from the user
 - Reverse the input
 - Print out the user input and the reversed version of the string

Output

```
Please type the original text:Cs114
```

```
The reversed text: 411sC
```

See [ReverseString.java](#)

Searching within a String: `indexOf`

`int indexOf(char ch)`

Returns the index of the first occurrence of `char ch` in this string.

Returns -1 if not matched.

`int indexOf(char ch, int fromIndex)`

Returns the index of the first occurrence of `char ch` in this string after `fromIndex`. Returns -1 if not matched.

`int indexOf(String s)`

Returns the index of the first occurrence of `String cs` in this string.

Returns -1 if not matched.

`int indexOf(String s, int fromIndex)`

Returns the index of the first occurrence of `String s` in this string after `fromIndex`. Returns -1 if not matched.

Searching within a String: `lastIndexOf`

`int lastIndexOf(char ch)`

Returns the index of the last occurrence of **`char ch`** in this string.

Returns -1 if not matched.

`int lastIndexOf(char ch, int fromIndex)`

Returns the index of the last occurrence of **`char ch`** in this string before **`fromIndex`**. Returns -1 if not matched.

`int lastIndexOf(String s)`

Returns the index of the last occurrence of **`String cs`** in this string.

Returns -1 if not matched.

`int lastIndexOf(String s, int fromIndex)`

Returns the index of the last occurrence of **`String s`** in this string before **`fromIndex`**. Returns -1 if not matched.

Searching within a string

0 3 5 9 11

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

"Welcome to Java".indexOf('o') returns 4.

"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('W') returns 0.

"Welcome to Java".lastIndexOf('o') returns 9.

"Welcome to Java".lastIndexOf('o', 5) returns 4.

"Welcome to Java".lastIndexOf("come") returns 3.

"Welcome to Java".lastIndexOf("Java", 5) returns -1.

"Welcome to Java".lastIndexOf("Java") returns 11.

String Class Example

- How would you replace a word within a String with another one? (Without using replace method)
- See [ReplaceOccurrence.java](#)

String Class Examples

- How would you replace a word within a String with another one? (Without using replace method)
- See [ReplaceOccurrence.java](#)
- Or you could use the replace method of String class:

```
str.replace(repFrom, repTo);
```

Outline

Using Classes and Objects

Creating Objects

The String Class

 **The Character Class**

The Random Class

The Math Classes

Formatting Output

Useful Methods in the Character Class

```
static boolean isDigit(char ch)
```

Determines if the specified character is a digit.

```
static boolean isLetter(char ch)
```

Determines if the specified character is a letter.

```
static boolean isLetterOrDigit(char ch)
```

Determines if the specified character is a letter or a digit.

```
static boolean isLowerCase(char ch)
```

Determines if the specified character is a lowercase letter.

```
static boolean isUpperCase(char ch)
```

Determines if the specified character is an uppercase letter.

```
static boolean isWhitespace(char ch)
```

Determines if the specified character is whitespace (spaces and tabs).

```
static char toLowerCase(char ch)
```

Converts `ch` to its lowercase equivalent, if any. If not, `ch` is returned

```
static char toUpperCase(char ch)
```

Converts `ch` to its uppercase equivalent, if any. If not, `ch` is returned unchanged.

Character Class

See [CharacterLowerCaseExample.java](#)

Character Class

```
public class CharacterLowerCaseExample {
    public static void main(String[] args)    {

        char c1 = 'A';
        char c2 = 'a';

        boolean b1 = Character.isLowerCase(c1);
        boolean b2 = Character.isLowerCase(c2);

        if(b1 == true){
            System.out.println(c1 + " is lowercase.");
        }
        else{
            System.out.println(c1 + " is not lowercase.");
        }
        if(b2 == true){
            System.out.println(c2 + " is lowercase.");
        }
        else{
            System.out.println(c2 + " is not lowercase.");
        }
    }
}
```

Outline

Using Classes and Objects

Creating Objects

The String Class

The Character Class

The Random Class

The Math Classes

Formatting Output

The Random Class

- The **Random** class is part of the `java.util` package
- It provides methods that generate pseudorandom numbers
- A **Random** object performs complicated calculations based on a *seed value* to produce a stream of seemingly random values
- See [RandomNumbers.java](#)

```
//*****
// RandomNumbers.java          Author: Lewis/Loftus
//
// Demonstrates the creation of pseudo-random numbers using the
// Random class.
//*****

import java.util.Random;

public class RandomNumbers
{
    //-----
    // Generates random numbers in various ranges.
    //-----
    public static void main (String[] args)
    {
        Random generator = new Random();
        int num1;
        float num2;

        num1 = generator.nextInt();
        System.out.println ("A random integer: " + num1);

        num1 = generator.nextInt(10);
        System.out.println ("From 0 to 9: " + num1);
    }
}
```

continued

continued

Sample Run

```
num1 A random integer: 672981683
Syst From 0 to 9: 0
      From 1 to 10: 3
num1 From 20 to 34: 30
Syst From -10 to 9: -4
num1 A random float (between 0-1): 0.18538326
Syst From 1 to 6: 3
```

```
num2 = generator.nextFloat();
System.out.println ("A random float (between 0-1): " + num2);

num2 = generator.nextFloat() * 6; // 0.0 to 5.999999
num1 = (int)num2 + 1;
System.out.println ("From 1 to 6: " + num1);
}
}
```

Quick Check

Given a `Random` object named `gen`, what range of values are produced by the following expressions?

`gen.nextInt(25)`

`gen.nextInt(6) + 1`

`gen.nextInt(100) + 10`

`gen.nextInt(50) + 100`

`gen.nextInt(10) - 5`

`gen.nextInt(22) + 12`

Quick Check

Given a `Random` object named `gen`, what range of values are produced by the following expressions?

	<u>Range</u>
<code>gen.nextInt(25)</code>	0 to 24
<code>gen.nextInt(6) + 1</code>	1 to 6
<code>gen.nextInt(100) + 10</code>	10 to 109
<code>gen.nextInt(50) + 100</code>	100 to 149
<code>gen.nextInt(10) - 5</code>	-5 to 4
<code>gen.nextInt(22) + 12</code>	12 to 33

Quick Check

Write an expression that produces a random integer in the following ranges:

Range

0 to 12

1 to 20

15 to 20

-10 to 0

Quick Check

Write an expression that produces a random integer in the following ranges:

Range

0 to 12	<code>gen.nextInt(13)</code>
1 to 20	<code>gen.nextInt(20) + 1</code>
15 to 20	<code>gen.nextInt(6) + 15</code>
-10 to 0	<code>gen.nextInt(11) - 10</code>

Outline

Using Classes and Objects

Creating Objects

The String Class

The Character Class

The Random Class



The Math Classes

Formatting Output

The Math Class

- The `Math` class is part of the `java.lang` package
- The `Math` class contains methods that perform various mathematical functions
- These include:
 - absolute value
 - square root
 - exponentiation
 - trigonometric functions

The Math Class

- The methods of the `Math` class are *static methods* (also called *class methods*)
- **Static methods are invoked through the class name – no object of the `Math` class is needed**

```
value = Math.cos(90) + Math.sqrt(delta);
```

- We discuss static methods further in Chapter 7
- See [Quadratic.java](#)

```

//*****
// Quadratic.java      Author: Lewis/Loftus
//
// Demonstrates the use of the Math class to perform a calculation
// based on user input.
//*****

import java.util.Scanner;

public class Quadratic
{
    //-----
    // Determines the roots of a quadratic equation.
    //-----
    public static void main (String[] args)
    {
        int a, b, c; // ax^2 + bx + c
        double discriminant, root1, root2;

        Scanner scan = new Scanner (System.in);

        System.out.print ("Enter the coefficient of x squared: ");
        a = scan.nextInt();

```

continued

continued

```
System.out.print ("Enter the coefficient of x: ");
b = scan.nextInt();

System.out.print ("Enter the constant: ");
c = scan.nextInt();

// Use the quadratic formula to compute the roots.
// Assumes a positive discriminant.

discriminant = Math.pow(b, 2) - (4 * a * c);
root1 = ((-1 * b) + Math.sqrt(discriminant)) / (2 * a);
root2 = ((-1 * b) - Math.sqrt(discriminant)) / (2 * a);

System.out.println ("Root #1: " + root1);
System.out.println ("Root #2: " + root2);
}
}
```

Useful Methods in the `Math` Class

<code>Math.abs(x)</code>	Returns the absolute value of x
<code>Math.min(x, y)</code>	Returns the smaller of x and y
<code>Math.max(x, y)</code>	Returns the larger of x and y
<code>Math.sqrt(x)</code>	Returns the square root of x
<code>Math.log(x)</code>	Returns the natural logarithm of x ($\log_e x$)
<code>Math.exp(x)</code>	Returns the inverse logarithm of x (e^x)
<code>Math.pow(x, y)</code>	Returns the value of x raised to the y power (x^y)
<code>Math.sin(theta)</code>	Returns the sine of $theta$, measured in radians
<code>Math.cos(theta)</code>	Returns the cosine of $theta$
<code>Math.tan(theta)</code>	Returns the tangent of $theta$
<code>Math.asin(x)</code>	Returns the angle whose sine is x
<code>Math.acos(x)</code>	Returns the angle whose cosine is x
<code>Math.atan(x)</code>	Returns the angle whose tangent is x
<code>Math.toRadians(degrees)</code>	Converts an angle from degrees to radians
<code>Math.toDegrees(radians)</code>	Converts an angle from radians to degrees

Math Example

- The following code provides examples on the use of most of the Math methods.
- Notice how we invoke the methods (without using a Math object)

HW: Study `MathExample.java`

Other Exercises

- See Converter.java

Outline

Using Classes and Objects

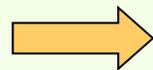
Creating Objects

The String Class

The Character Class

The Random Class

The Math Classes



Formatting Output

Formatting Output

- It is often necessary to format output values in certain ways so that they can be presented properly
- The Java standard class library contains classes that provide formatting capabilities
- The `NumberFormat` class allows you to format values as currency or percentages
- The `DecimalFormat` class allows you to format values based on a pattern
- Both are part of the `java.text` package

Formatting Output

- The `NumberFormat` class has static methods that return a formatter object

```
getCurrencyInstance()
```

```
getPercentInstance()
```

- Each formatter object has a method called `format` that returns a string with the specified information in the appropriate format
- See [Purchase.java](#)

```

//*****
// Purchase.java          Author: Lewis/Loftus
//
// Demonstrates the use of the NumberFormat class to format output.
//*****

import java.util.Scanner;
import java.text.NumberFormat;

public class Purchase
{
    //-----
    // Calculates the final price of a purchased item using values
    // entered by the user.
    //-----
    public static void main (String[] args)
    {
        final double TAX_RATE = 0.06; // 6% sales tax

        int quantity;
        double subtotal, tax, totalCost, unitPrice;

        Scanner scan = new Scanner (System.in);

```

continued

continued

```
NumberFormat fmt1 = NumberFormat.getCurrencyInstance();
NumberFormat fmt2 = NumberFormat.getPercentInstance();

System.out.print ("Enter the quantity: ");
quantity = scan.nextInt();

System.out.print ("Enter the unit price: ");
unitPrice = scan.nextDouble();

subtotal = quantity * unitPrice;
tax = subtotal * TAX_RATE;
totalCost = subtotal + tax;

// Print output with appropriate formatting
System.out.println ("Subtotal: " + fmt1.format(subtotal));
System.out.println ("Tax: " + fmt1.format(tax) + " at "
                    + fmt2.format(TAX_RATE));
System.out.println ("Total: " + fmt1.format(totalCost));
}
}
```

continued

Sample Run

```
NumberFormat f
NumberFormat f
System.out.print
quantity = sca
Enter the quantity: 5
Enter the unit price: 3.87
Subtotal: $19.35
Tax: $1.16 at 6%
Total: $20.51
tance();
ance();

System.out.print ("Enter the unit price: ");
unitPrice = scan.nextDouble();

subtotal = quantity * unitPrice;
tax = subtotal * TAX_RATE;
totalCost = subtotal + tax;

// Print output with appropriate formatting
System.out.println ("Subtotal: " + fmt1.format(subtotal));
System.out.println ("Tax: " + fmt1.format(tax) + " at "
                    + fmt2.format(TAX_RATE));
System.out.println ("Total: " + fmt1.format(totalCost));
}
}
```

Formatting Output

- The `DecimalFormat` class can be used to format a floating point value in various ways
- For example, you can specify that the number should be truncated to three decimal places
- The constructor of the `DecimalFormat` class takes a string that represents a pattern for the formatted number
- See [CircleStats.java](#)

```

//*****
// CircleStats.java      Author: Lewis/Loftus
//
// Demonstrates the formatting of decimal values using the
// DecimalFormat class.
//*****

import java.util.Scanner;
import java.text.DecimalFormat;

public class CircleStats
{
    //-----
    // Calculates the area and circumference of a circle given its
    // radius.
    //-----
    public static void main (String[] args)
    {
        int radius;
        double area, circumference;

        Scanner scan = new Scanner (System.in);

```

continued

continued

```
System.out.print ("Enter the circle's radius: ");
radius = scan.nextInt();

area = Math.PI * Math.pow(radius, 2);
circumference = 2 * Math.PI * radius;

// Round the output to three decimal places
DecimalFormat fmt = new DecimalFormat ("0.###");

System.out.println ("The circle's area: " + fmt.format(area));
System.out.println ("The circle's circumference: "
                    + fmt.format(circumference));
}
}
```

Sample Run

continued

```
System.out.println("Enter the circle's radius: ");
radius = 5;
System.out.println("The circle's area: " + area);
System.out.println("The circle's circumference: " + circumference);
```

```
area = Math.PI * Math.pow(radius, 2);
circumference = 2 * Math.PI * radius;
```

```
// Round the output to three decimal places
```

```
DecimalFormat fmt = new DecimalFormat("0.###");
```

```
System.out.println("The circle's area: " + fmt.format(area));
```

```
System.out.println("The circle's circumference: "
    + fmt.format(circumference));
```

```
}
```

```
}
```

Using Locale class

- What if you want to display Euro symbol, but not the default money symbol on your computer?
- The answer is to use a country that uses Euro as the currency symbol as the parameter for the `getCurrencyInstance` method of the `NumberFormat` class.
- You can also generate your own `Locale` like `tr` in the following example
- See [EuroSymbol.java](#)

Using useLocale() method

- Have you ever encountered the run-time error, InputMismatchException when you have tried to input a floating-point number with an incorrect decimal point symbol?
- Would you not prefer to make sure that you can use the . as the decimal point ?
- See [ReadingInUSLocale.java](#)