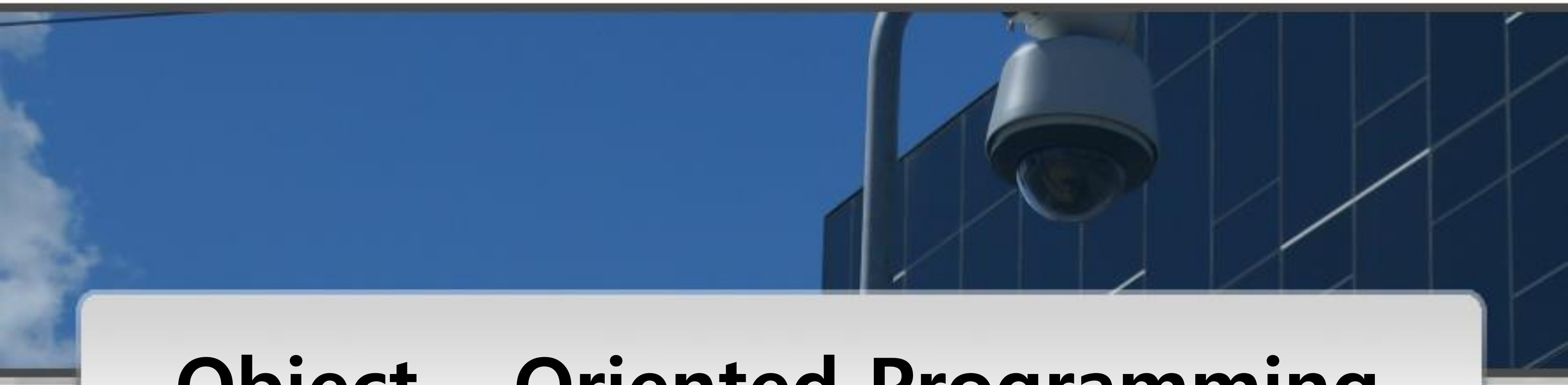


Object – Oriented Programming

Lab #01

COMPUTER SECURITY LABORATORY

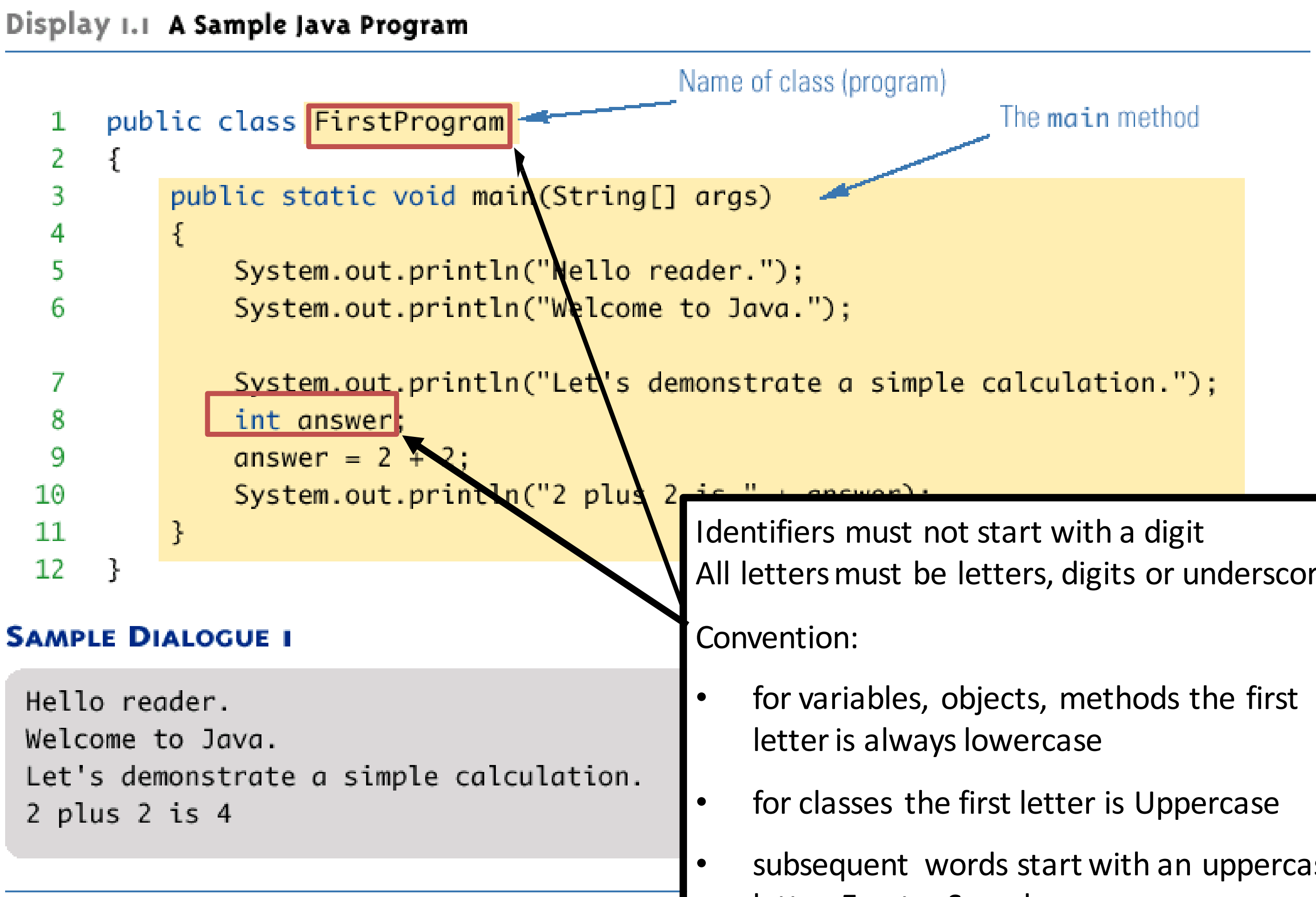
HANYANG UNIVERSITY

Contents

Java Language

- **Java is an Object Oriented Programming language (OOP)**
  - *Objects can perform actions*
  - *Objects are affected by the actions of other objects*
  - *The actions of Objects are known as* ***methods***
  - *Same types of Objects are said to be in the same Class*
- **Java Applications**
  - Java classes with a **main** method

Java Language (Contd)



Display 1.1 **A Sample Java Program**

Name of class (program)
The main method

```
public class FirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("Hello reader.");
        System.out.println("Welcome to Java.");

        System.out.println("Let's demonstrate a simple calculation.");
        int answer;
        answer = 2 + 2;
        System.out.println("2 plus 2 is " + answer);
    }
}
```

Sample Dialogue 1

```
Hello reader.
Welcome to Java.
Let's demonstrate a simple calculation.
2 plus 2 is 4
```

Identifiers must not start with a digit
All letters must be letters, digits or underscore

Convention:

- for variables, objects, methods the first letter is always lowercase
- for classes the first letter is Uppercase
- subsequent words start with an uppercase letter. E.g. topSpeed

Identifiers

- Identifiers are names for variables, classes, etc.
- Must not start with a digit, all the characters must be letters, digits or the underscore symbol
- Good ones are compact, but indicate what they stand for
  - radius, width, height, length
- Bad ones are either too long
  - theRadiusOfTheCircle
  - theWidthOfTheBoxThatIsBeingUsed
  - the_width_of_the_box_that_is_being_used
- Or too short
  - a, b, c, d, e
- Good identifiers will help understand your program!

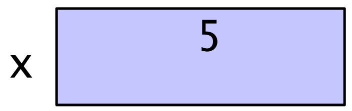
Keywords

- **Some words are reserved, and can't be used as identifiers**

```
// Purpose: display a quotation in a console window

public class DisplayForecast {

  // method main(): application entry point
  public static void main(String[] args) {
     System.out.print("I think there is a world market for");
     System.out.println(" maybe five computers.");
     System.out.println("       Thomas Watson, IBM,1943.");
  }
}
```

Variables

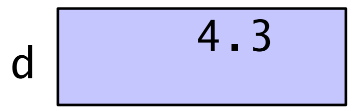
- **We've seen variables before in math**
  - *y* = *mx* + *b*
  - Here *y*, *m*, *x*, and *b* can hold any value
- **To store things in a computer program, we also use variables**
- **Example:**
  - int x = 5;
  - This defines/declares an integer variable with value 5



- **The variable is x**
- **The type is int**

Variables (Contd)

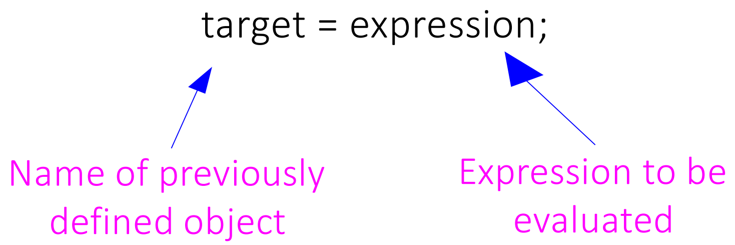
- **An integer variable can only hold integers**
  - In other words, it <u>can't</u> hold 4.3
- **To hold float point values, we use the double type**
  - double d = 4.3;
- **Same with**
  - double d;
  - d = 4.3;



- **The variable is d**
- **The type is double**

Variables (Contd)

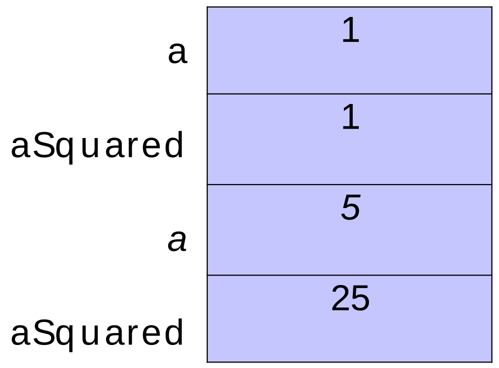
- **Assignment operator =**
  - Allows the variable to be updated



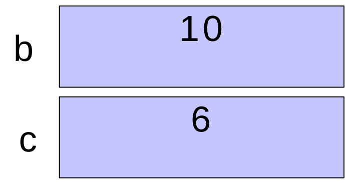
- **Consider**
  - int j = 11;
  - j = 1985;

Primitive variable assignment

- Consider
  - int a = 1;
    int aSquared = a * a;
    a = 5;
    aSquared = a * a;



- Consider
  - int b = 10;
    int c = b – 4;



Operators +=, -=, *=, ++, --

int x = 15;			x	15
x += 5;	⇨	x = x + 5;	x	20
x -= 10;	⇨	x = x - 10;	x	10
x *= 5;	⇨	x = x * 5;	x	50
x++;	⇨	x = x + 1;	x	51
x--;	⇨	x = x - 1;	x	50

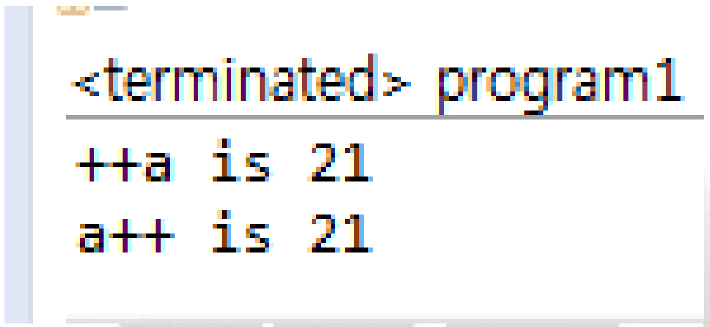
Increment/Decrement Operations

- You can do a pre-increment, or a post-increment

Pre	Post
++x;	x++;
--x;	x--;

- Difference?

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

    int a=20;
    System.out.println("++a is " + ++a );
    System.out.println("a++ is " + a++ );

}
```



```
<terminated> program1
++a is 21
a++ is 21
```

Printing Variables

- **To print a variable to the screen, put it in a System.out.println() statement:**
  - int x = 5;
  - System.out.println("The value of x is " + x);

- **Important points:**
  - Strings are enclosed in double quotes
  - If there are multiple parts to be printed, they are separated by a plus sign. This will automatically concatenate the parts.

Variables must be declared before use

- The following code will not work:
  - x = 5;
  - System.out.println(x);

- Java requires you to declare x before you use it

You can only declare variables once

- The following code will not work:
  - int x = 5;
  - int x = 6;
- Java can have only one variable named x
  - So you can't declare multiple variables with the same name

Variable Initialization

- Consider the following code:
  - int x;
  - System.out.println(x);
- What happens?
- Error message:
  - variable x might not have been initialized
- Java also requires you to give x a value before you use it

Primitive variable types

- **Java has 8 primitive types:**

– float – double	floating point numbers
– boolean	two values: true and false
– char	a single character
– byte – short – int – long	integer numbers

Primitive real (floating-point) types

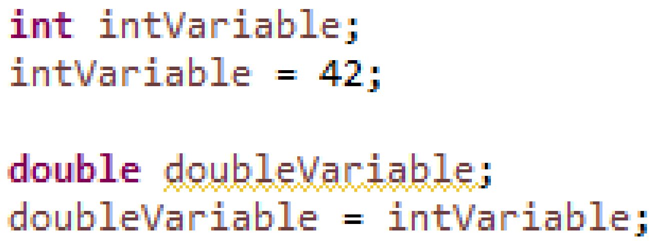
- A float takes up 4 bytes of space
  - Has 6 decimal places of accuracy: 3.14159
- A double takes up 8 bytes of space
  - Has 15 decimal places of accuracy: 3.14159263538979

Primitive integer types

Type	Bytes	Minimum value	Maximum value
byte	1	-2^{7}=-128	$2^{7}-1$=127
short	2	-2^{15}= -32,768	$2^{15}-1$= 32,767
int	4	-2^{31}=-2,147,483,648	$2^{31}-1$=2,147,483,647
long	8	-2^{63}=-9,223,372,036, 854,775,808	$2^{63}-1$=9,223,372,036, 854,775,807

Type Casting

- **Consider the code. Is it possible to assign the value of a variable of one type to a variable of another type?**



```
int intVariable;
intVariable = 42;

double doubleVariable;
doubleVariable = intVariable;
```

- **Yes**
  - You can assign the value of any type to that of another type that is lower in the following list:
  - byte < short < int < long < float < double

Primitive Boolean type

- The boolean type has only two values:
  - true
  - false

- There are boolean-specific operators
  - && is and
  - || is or
  - ! is not

Constants

- Consider the following:
  - final int x = 5;
- The value of x can NEVER be changed!
  - The value assigned to it is “final”
- This is how Java defines constants
- Constants have a specific naming scheme
  - MILES_PER_KILOMETER
  - All caps, with underscores for spaces

Expressions

- What is the value used to initialize expression
  - int expression = 4 + 2 * 5;
- What value is displayed
  - System.out.println(5 / 2.0);
- Java rules in a nutshell
  - Each operator has a precedence level and an associativity
    - Operators with higher precedence are done first
      - * and / have higher precedence than + and -
  - When floating-point is used, the result is floating point

Question on expressions

- Does the following statement compute the average of double variables a, b, and c? Why or why not?
  - double average = a + b + c / 3.0;

Java operators

- **The following are the common operators for ints:**
  - + - / * %
  - Division /
    - 6 / 2 yields 3
    - 7 / 2 yields 3, not 3.5
    - Because everything is an int, the answer is an int
  - Modulus is %
    - Returns the remainder
    - 7 % 2 yields 1
    - 6 % 2 yields 0
- **Floats and doubles use the same first four operators**
  - + - / *
  - 7.0 / 2.0 yields 3.5
  - 7.0 / 2 yields 3.5
  - 7 / 2.0 yields 3.5
  - 7 / 2 yields 3

Java operators (Contd)

- Booleans have their own operators
  - && is AND
    - Only true when both operands are true
    - true && true yields true
    - false && true yields false
  - || is OR
    - True when either of the operands (or both) are true
    - true || false yields true
    - false || false yields false
  - ! is NOT
    - Changes the value
    - !true yields false
    - !false yields true

Self-Test (1)

- *counter*와 *totalDistance* 2개의 변수를 선언할 것
- *counter*는 int type이며 3으로 초기화 되고,
  *totalDistance*는 int type이며 15로 초기화 된다.
- *quotient*와 *remainder* 2개의 변수를 int type으로 선언할 것
- *quotient*은 *totalDistance*를 *counter*로 나눴을 때의 몫
  *remainder*는 *totalDistance*를 *counter*로 나눴을 때의 나머지 일 때
  *quotient*과 *remainder*를 출력하는 프로그램을 작성할 것

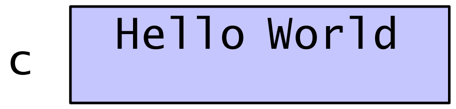
The String Class

- **The String class is used to store and process strings of characters**
  - A string is a sequence of characters gathered together, like the word “Hello”, or the phrase “practice makes perfect”
  - String creation:
    - String s = “Java is fun.”;
    - String s = new String(“Java is fun.”);

0	1	2	3	4	5	6	7	8	9	10	11
J	a	v	a		i	s		f	u	n	.

Working with Strings

- A convenience is in String concatenation:
  - String a = “hello “;
    String b = “world”;
    String c;
    c = a + b;

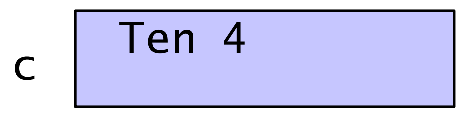


- This returns a new string c that is a with b added to it at the end The + operator is widely used in print statements
- We accept that we’re not arithmetically adding these two Strings.

Working with Strings (Contd)

- **Take the following example, where we are concatenating a String `a` with an int `i`:**

  -
    ```
    String a = "Ten ";
    int i = 4;
    String c;

    c = a + i;
    ```



Working with Strings (Contd)

- Anytime you concatenate a String with another type, the other type is first converted into a String and then the two are concatenated. Really, this is done using method `toString()` which every object inherits from `Object`

```
– c = a + i; // c = a + i.toString();
```

Working with Strings (Contd)

- String a = ""
  int b = 2;
  int c = 3;

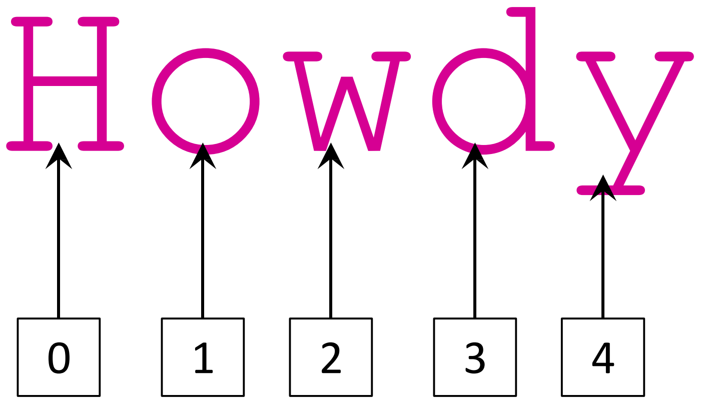
- **The above statement is a Syntax error. To force the conversion to String, at least one of the operands on the + sign must be a String.**

Sub-Strings (1)

- You can extract a substring from a larger String object with the `substring()` method of class `String`.
  - String greet = “Howdy”;
    String s = greet.substring(0, 4)

The first argument 0, is the 1st character of the substring that you do want to copy.

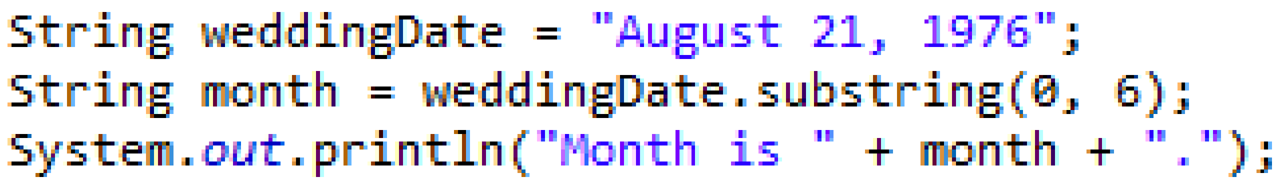
The 4 is the 1st character that you don’t want to copy.



- So… s is equal to “howd”

Sub-Strings (2)

- Consider



```
String weddingDate = "August 21, 1976";
String month = weddingDate.substring(0, 6);
System.out.println("Month is " + month + ".");
```

- What is output?
  - Month is August

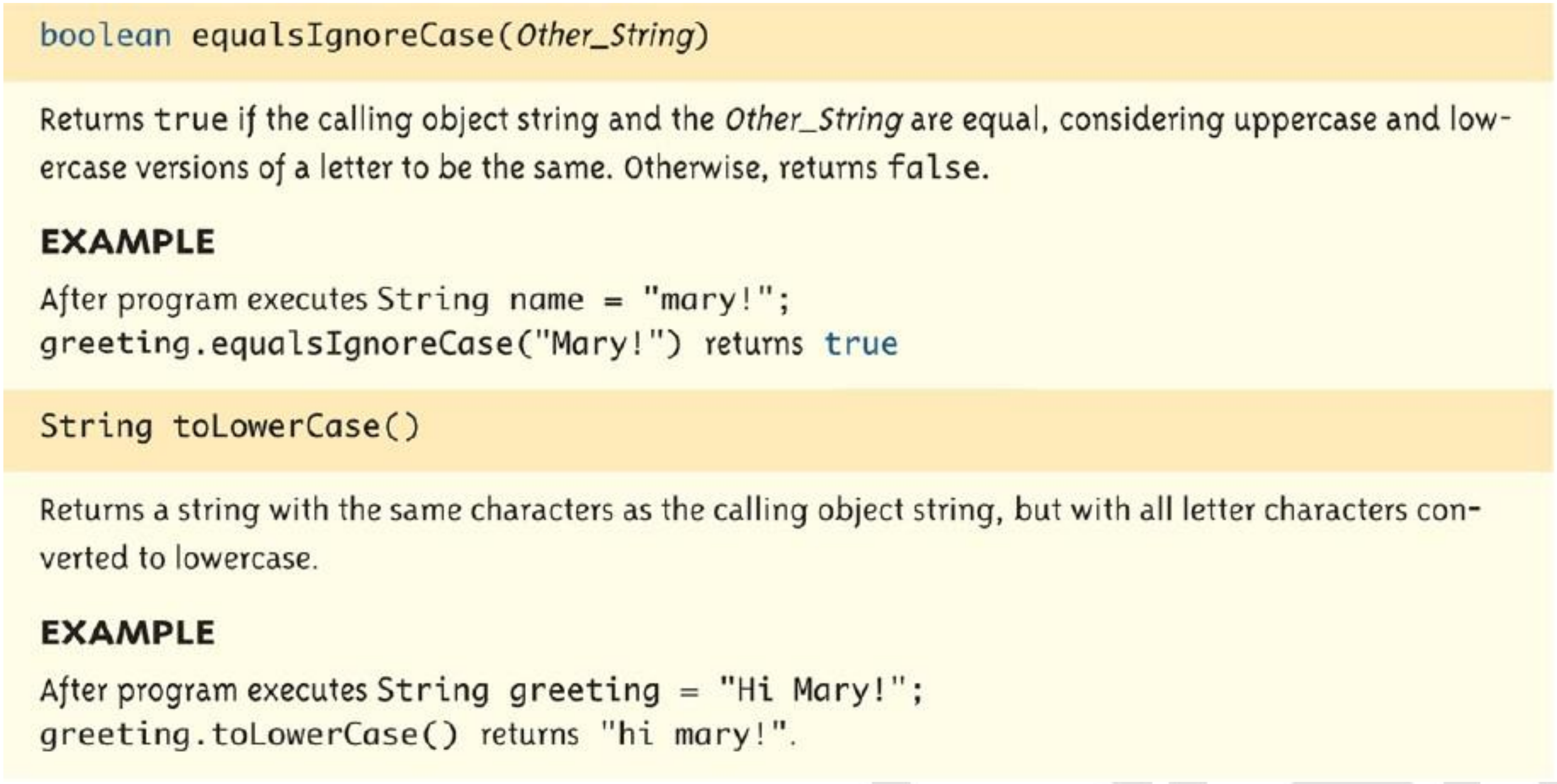
Replacing Substrings

- **This returns a string with the specified substrings replaced**

  -
    ```
    String b = "hacker heaven";
    String n = "";

    n = b.replace('h', 'H');
    ```

- **Note: This does not change the original, It returns a new String object in which the changes have been made.**

Some more methods



`boolean equalsIgnoreCase(`*`Other_String`*`)`

Returns `true` if the calling object string and the *Other_String* are equal, considering uppercase and lowercase versions of a letter to be the same. Otherwise, returns `false`.

EXAMPLE

After program executes `String name = "mary!";`
`greeting.equalsIgnoreCase("Mary!")` returns `true`

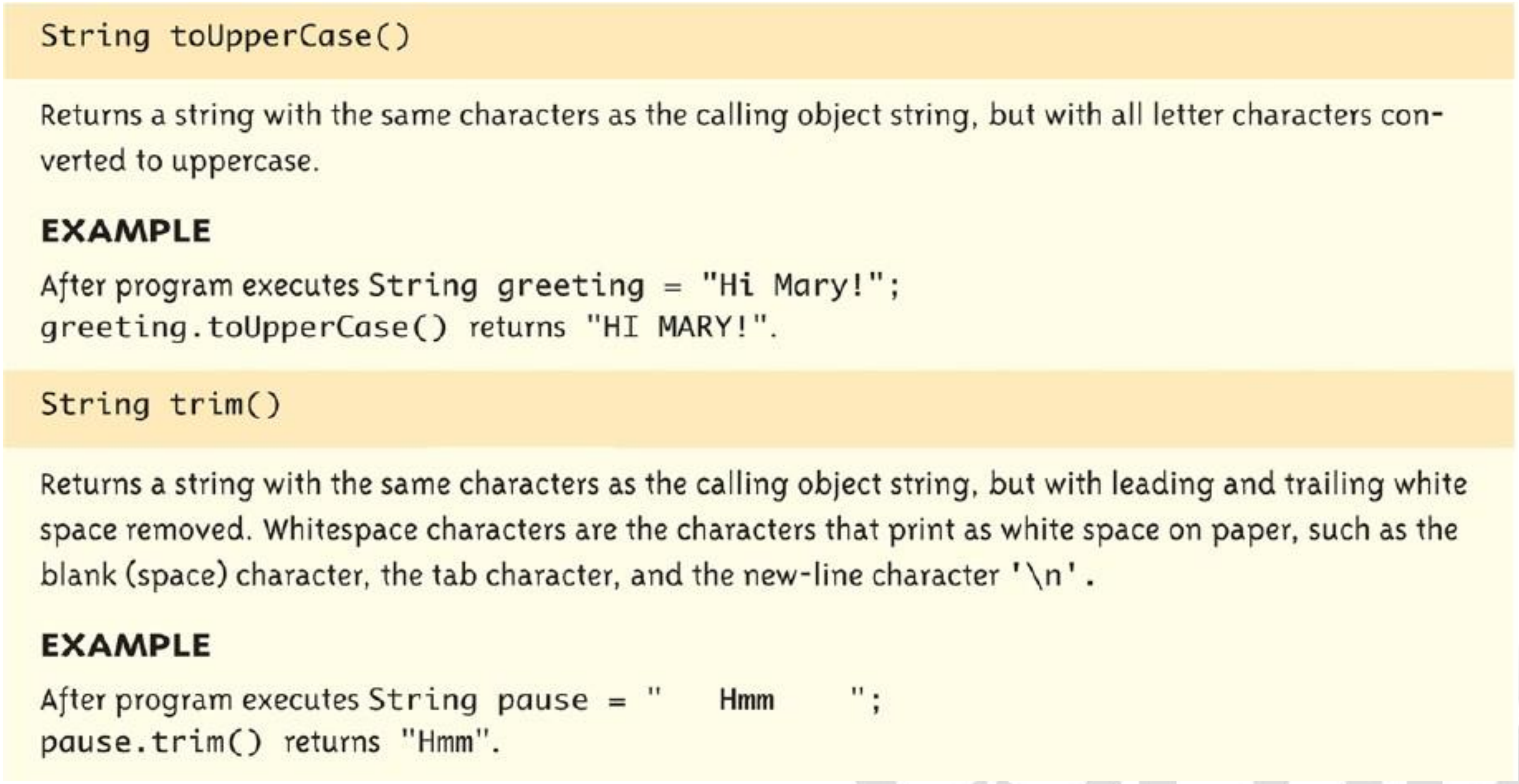
`String toLowerCase()`

Returns a string with the same characters as the calling object string, but with all letter characters converted to lowercase.

EXAMPLE

After program executes `String greeting = "Hi Mary!";`
`greeting.toLowerCase()` returns `"hi mary!"`.

Some more methods (Contd)



`String toUpperCase()`

Returns a string with the same characters as the calling object string, but with all letter characters converted to uppercase.

EXAMPLE

After program executes `String greeting = "Hi Mary!";`
`greeting.toUpperCase()` returns `"HI MARY!"`.

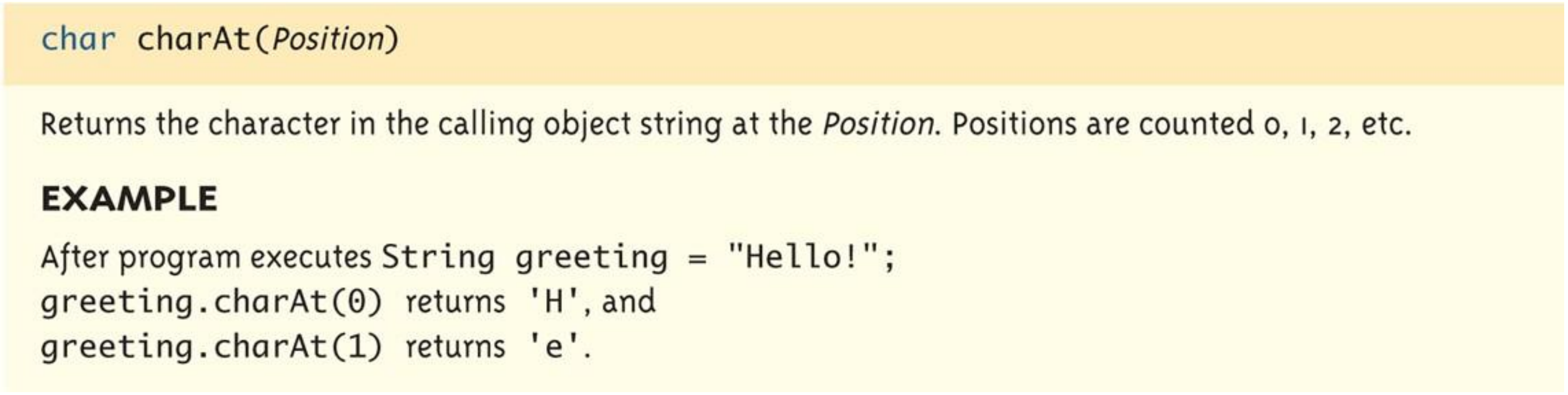
`String trim()`

Returns a string with the same characters as the calling object string, but with leading and trailing white space removed. Whitespace characters are the characters that print as white space on paper, such as the blank (space) character, the tab character, and the new-line character `'\n'`.

EXAMPLE

After program executes `String pause = "    Hmm     ";`
`pause.trim()` returns `"Hmm"`.

Some more methods (Contd)

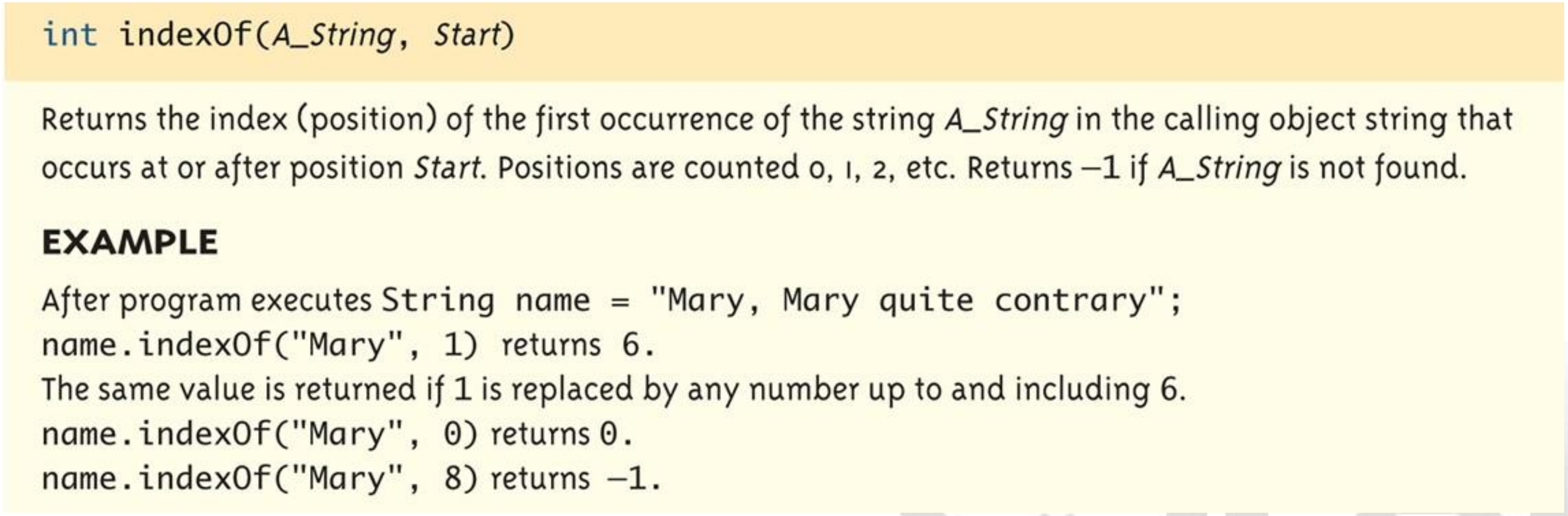


`char charAt(`*`Position`*`)`

Returns the character in the calling object string at the *Position*. Positions are counted 0, 1, 2, etc.

EXAMPLE

After program executes `String greeting = "Hello!";`
`greeting.charAt(0)` returns `'H'`, and
`greeting.charAt(1)` returns `'e'`.



`int indexOf(`*`A_String`*`, `*`Start`*`)`

Returns the index (position) of the first occurrence of the string *A_String* in the calling object string that occurs at or after position *Start*. Positions are counted 0, 1, 2, etc. Returns −1 if *A_String* is not found.

EXAMPLE

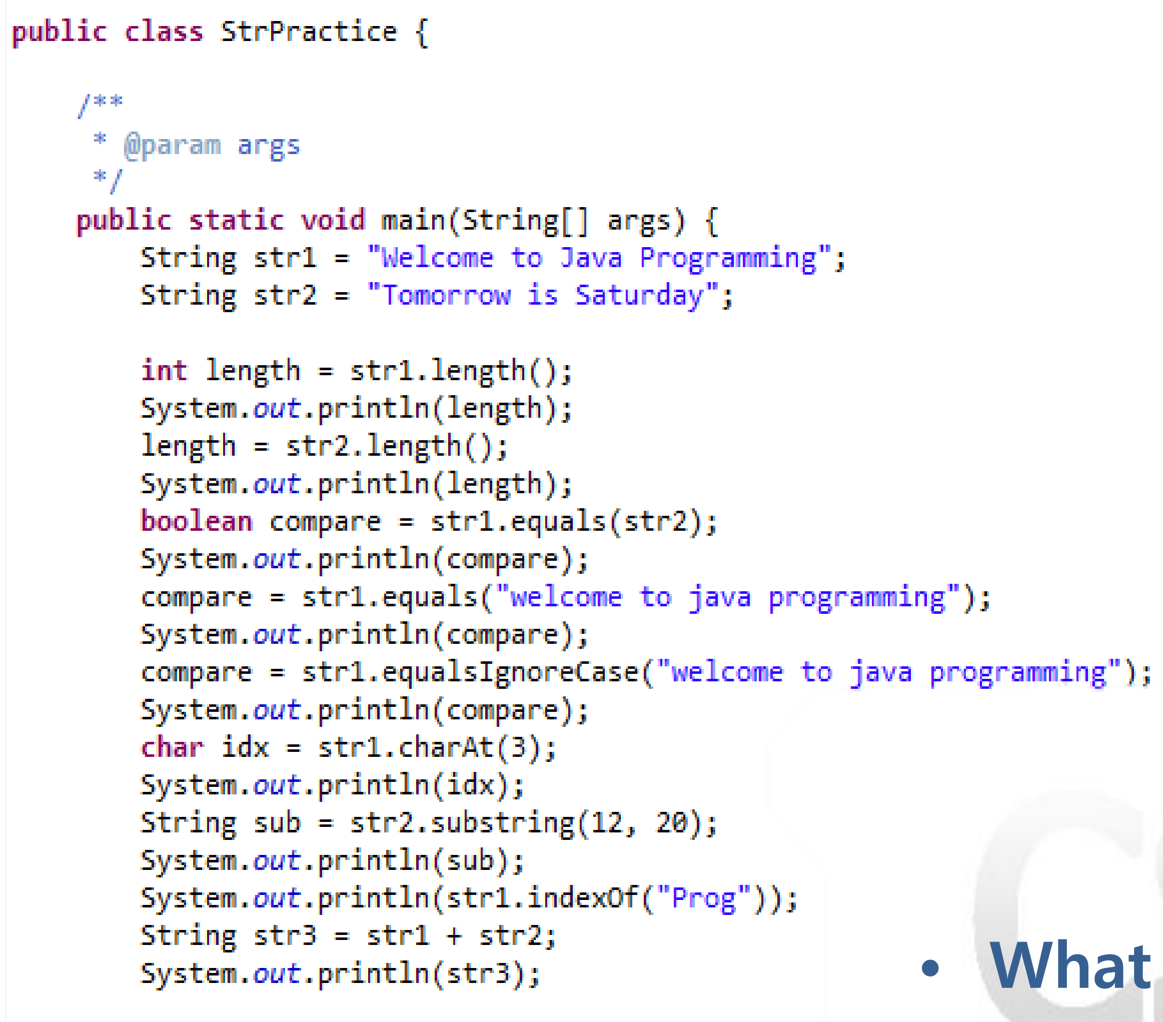
After program executes `String name = "Mary, Mary quite contrary";`
`name.indexOf("Mary", 1)` returns `6`.
The same value is returned if 1 is replaced by any number up to and including 6.
`name.indexOf("Mary", 0)` returns `0`.
`name.indexOf("Mary", 8)` returns `–1`.

String method examples

```
// index 012345678901
String s1 = “Stuart Reges”;
String s2 = “Marty Stepp”;

System.out.println(s1.length()); // 12
System.out.println(s1.indexOf("e")); // 8
System.out.println(s1.substring(7, 10)); // “Reg”
String s3 = s2.substring(1, 7);
System.out.println(s3.toLowerCase()); // “arty s”
```

- **Given the following string:**
  - String book = “Hello Java Programs";
  - How would you extract the word “Java”?

String method examples (Contd)

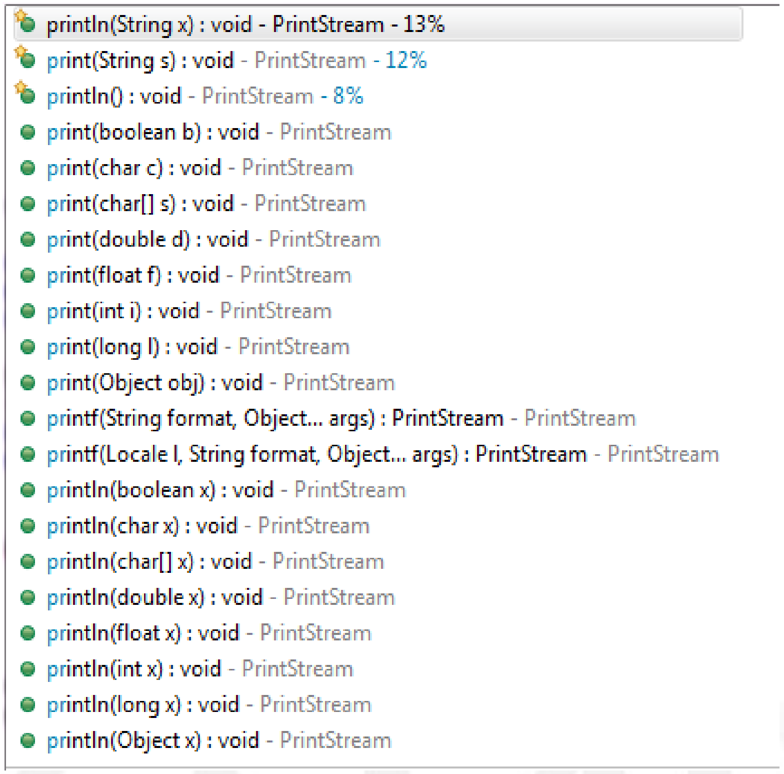


```java
public class StrPractice {

    /**
     * @param args
     */
    public static void main(String[] args) {
        String str1 = "Welcome to Java Programming";
        String str2 = "Tomorrow is Saturday";

        int length = str1.length();
        System.out.println(length);
        length = str2.length();
        System.out.println(length);
        boolean compare = str1.equals(str2);
        System.out.println(compare);
        compare = str1.equals("welcome to java programming");
        System.out.println(compare);
        compare = str1.equalsIgnoreCase("welcome to java programming");
        System.out.println(compare);
        char idx = str1.charAt(3);
        System.out.println(idx);
        String sub = str2.substring(12, 20);
        System.out.println(sub);
        System.out.println(str1.indexOf("Prog"));
        String str3 = str1 + str2;
        System.out.println(str3);
```

- What will be the output?

Console Output

- Output to the console can be performed using the System.out.println(<*arguments*>)

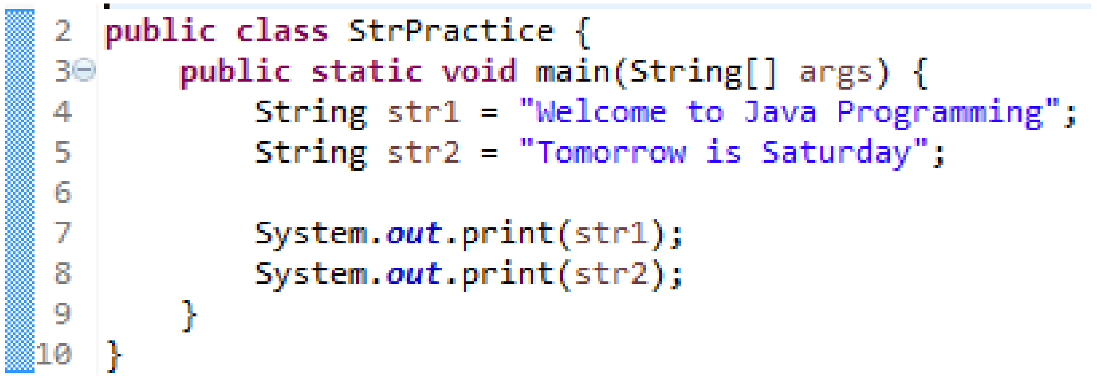


```
println(String x) : void - PrintStream - 13%
print(String s) : void - PrintStream - 12%
println() : void - PrintStream - 8%
print(boolean b) : void - PrintStream
print(char c) : void - PrintStream
print(char[] s) : void - PrintStream
print(double d) : void - PrintStream
print(float f) : void - PrintStream
print(int i) : void - PrintStream
print(long l) : void - PrintStream
print(Object obj) : void - PrintStream
printf(String format, Object... args) : PrintStream - PrintStream
printf(Locale l, String format, Object... args) : PrintStream - PrintStream
println(boolean x) : void - PrintStream
println(char x) : void - PrintStream
println(char[] x) : void - PrintStream
println(double x) : void - PrintStream
println(float x) : void - PrintStream
println(int x) : void - PrintStream
println(long x) : void - PrintStream
println(Object x) : void - PrintStream
```

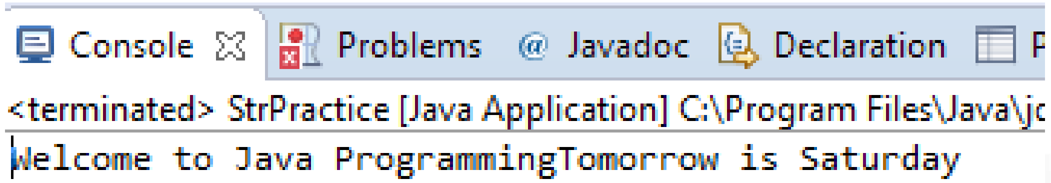
- There are various methods associated with printing.
  - Print()
  - Println()
  - Printf()
- Each accepting different parameters

Print vs. Println

- If the *print* method is used, then the cursor is *not moved* to a newline.



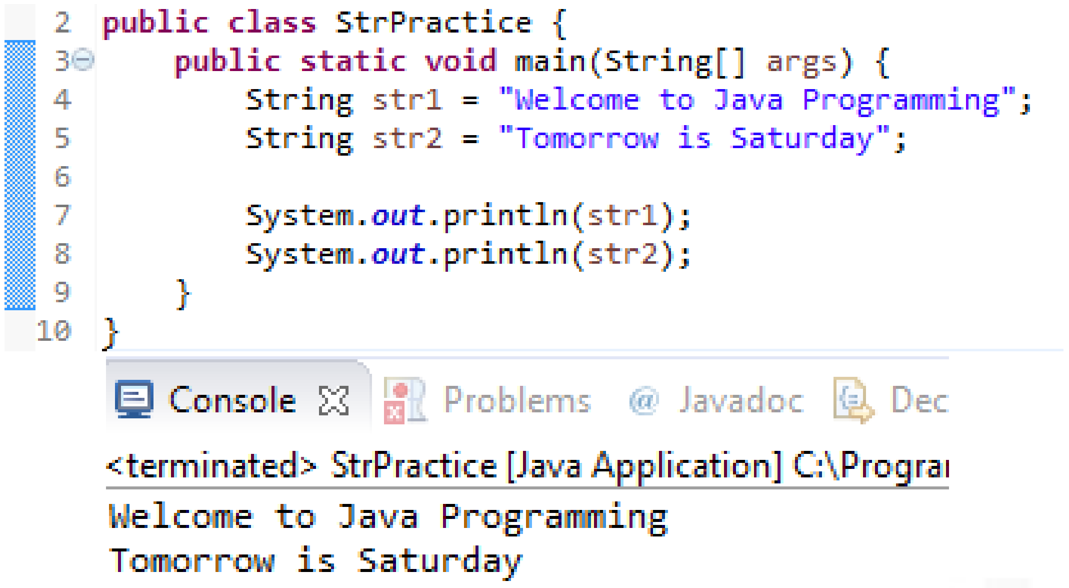
```java
public class StrPractice {
    public static void main(String[] args) {
        String str1 = "Welcome to Java Programming";
        String str2 = "Tomorrow is Saturday";

        System.out.print(str1);
        System.out.print(str2);
    }
}
```



```
<terminated> StrPractice [Java Application] C:\Program Files\Java\jc
Welcome to Java ProgrammingTomorrow is Saturday
```

Print vs. Println (Contd)

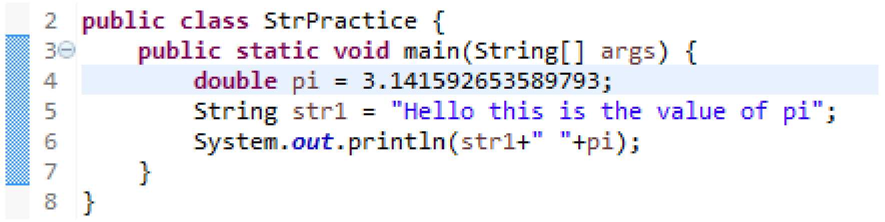
- If the *println* method is used then the cursor is *moved* to a new line



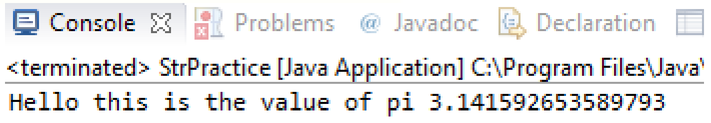
```
public class StrPractice {
    public static void main(String[] args) {
        String str1 = "Welcome to Java Programming";
        String str2 = "Tomorrow is Saturday";

        System.out.println(str1);
        System.out.println(str2);
    }
}
```

```
<terminated> StrPractice [Java Application] C:\Progra
Welcome to Java Programming
Tomorrow is Saturday
```

Formatting Output



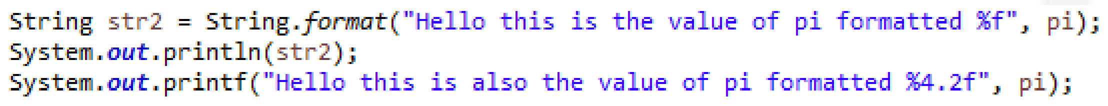
```
public class StrPractice {
    public static void main(String[] args) {
        double pi = 3.141592653589793;
        String str1 = "Hello this is the value of pi";
        System.out.println(str1+" "+pi);
    }
}
```



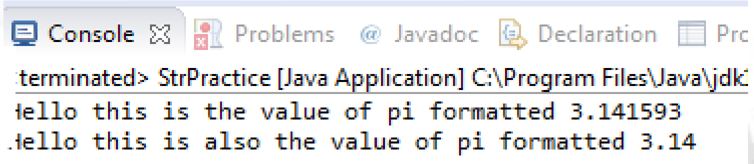
```
<terminated> StrPractice [Java Application] C:\Program Files\Java\
Hello this is the value of pi 3.141592653589793
```

- **The output of the code is the entire float given to 15 decimal places**

- **Java allows us to format out output using one of two ways**
  - Format() method of the string class.
  - Printf() method of the System.out object.



```
String str2 = String.format("Hello this is the value of pi formatted %f", pi);
System.out.println(str2);
System.out.printf("Hello this is also the value of pi formatted %4.2f", pi);
```



```
terminated> StrPractice [Java Application] C:\Program Files\Java\jdk
Hello this is the value of pi formatted 3.141593
Hello this is also the value of pi formatted 3.14
```

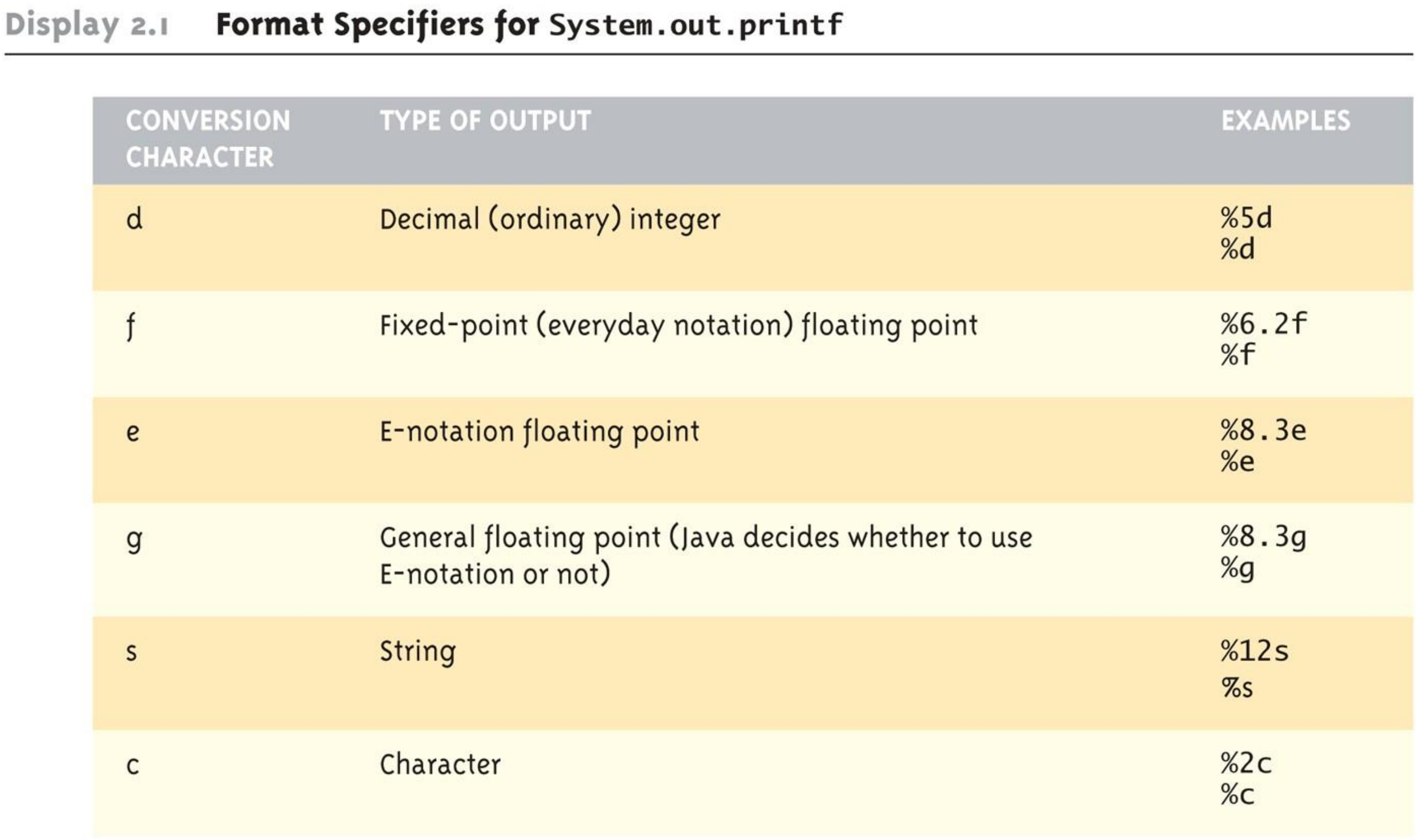
Formatting Output (Contd)

- **If we analyze the arguments of the methods then**
  1. The first argument is the *format string*
  2. The other arguments are the variables or values to be output
  3. The number of variables/values must correspond to the number of format specifiers in the format string.

- **The following will result in an error**

```
System.out.printf("Hello this is also the value of pi %f formatted %4.2f", pi);
```

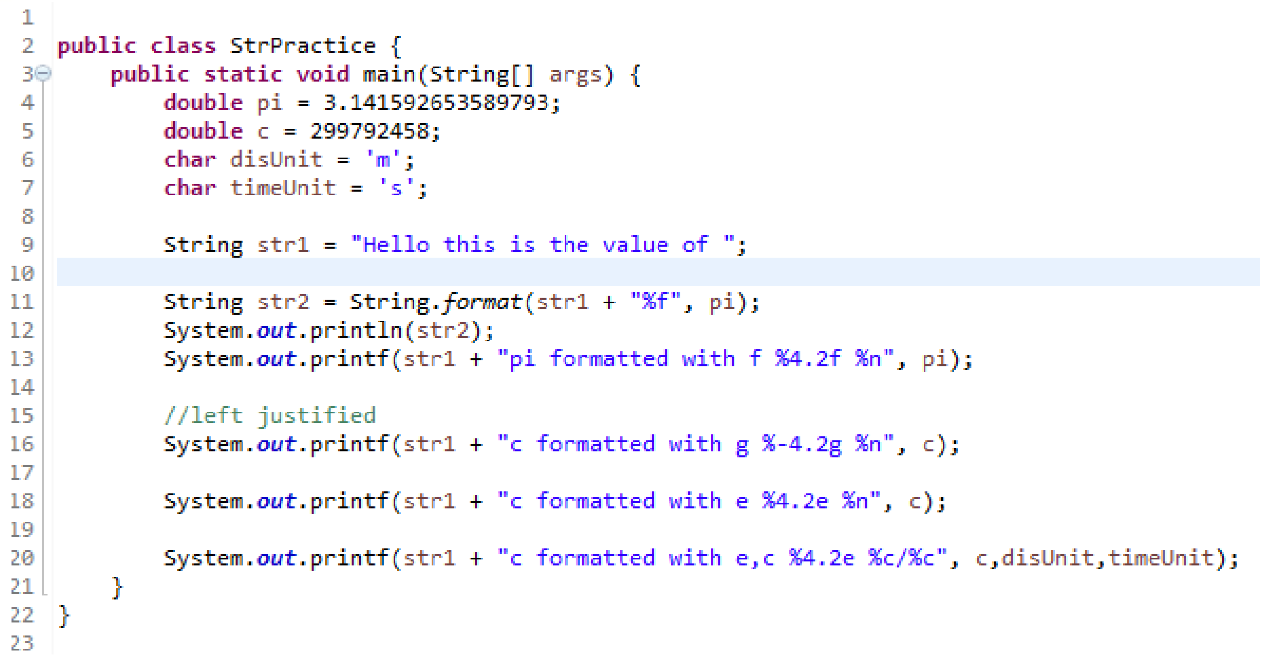
Formatted specifiers



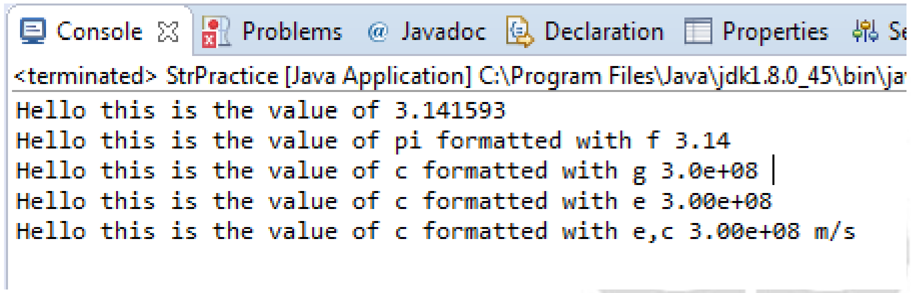
Display 2.1 **Format Specifiers for `System.out.printf`**

CONVERSION CHARACTER	TYPE OF OUTPUT	EXAMPLES
d	Decimal (ordinary) integer	`%5d` `%d`
f	Fixed-point (everyday notation) floating point	`%6.2f` `%f`
e	E-notation floating point	`%8.3e` `%e`
g	General floating point (Java decides whether to use E-notation or not)	`%8.3g` `%g`
s	String	`%12s` `%s`
c	Character	`%2c` `%c`

Formatting Output (Contd)



```java

public class StrPractice {
    public static void main(String[] args) {
        double pi = 3.141592653589793;
        double c = 299792458;
        char disUnit = 'm';
        char timeUnit = 's';

        String str1 = "Hello this is the value of ";

        String str2 = String.format(str1 + "%f", pi);
        System.out.println(str2);
        System.out.printf(str1 + "pi formatted with f %4.2f %n", pi);

        //left justified
        System.out.printf(str1 + "c formatted with g %-4.2g %n", c);

        System.out.printf(str1 + "c formatted with e %4.2e %n", c);

        System.out.printf(str1 + "c formatted with e,c %4.2e %c/%c", c,disUnit,timeUnit);
    }
}

```



```
<terminated> StrPractice [Java Application] C:\Program Files\Java\jdk1.8.0_45\bin\ja
Hello this is the value of 3.141593
Hello this is the value of pi formatted with f 3.14
Hello this is the value of c formatted with g 3.0e+08
Hello this is the value of c formatted with e 3.00e+08
Hello this is the value of c formatted with e,c 3.00e+08 m/s
```

Console Input

- It is just as important to input information into your programs as it is to output information.

- Prior to Java 5.0 there was no simple method to read data from the keyboard
  - Programmers needed to create a BufferedReader
  - Programmers also had to be aware of and handle exceptions that may occur.

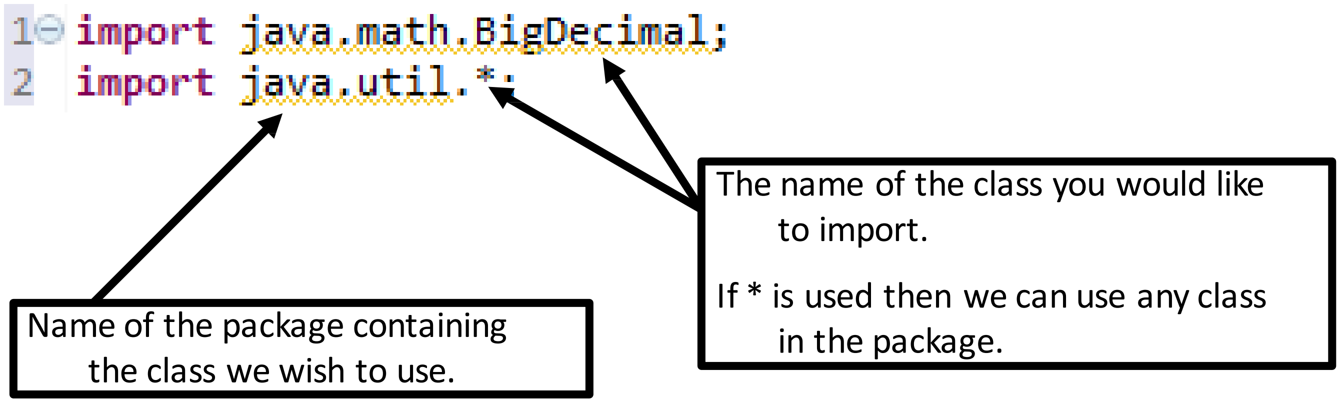
Console Input (Contd)

- Java allows for keyboard input using the Scanner class.
- Scanner is a part of the java.util package

- Before you can use the Scanner class you must first import that package.

Importing Packages

- **Packages are collections of "pre-compiled" classes that can be utilized.**
- **We must import the package containing the class we would like to use before we can use that class.**
  - This is done by using the keyword *import* and the *package name* at the start of you code.



```
import java.math.BigDecimal;
import java.util.*;
```

Name of the package containing the class we wish to use.

The name of the class you would like to import.

If * is used then we can use any class in the package.

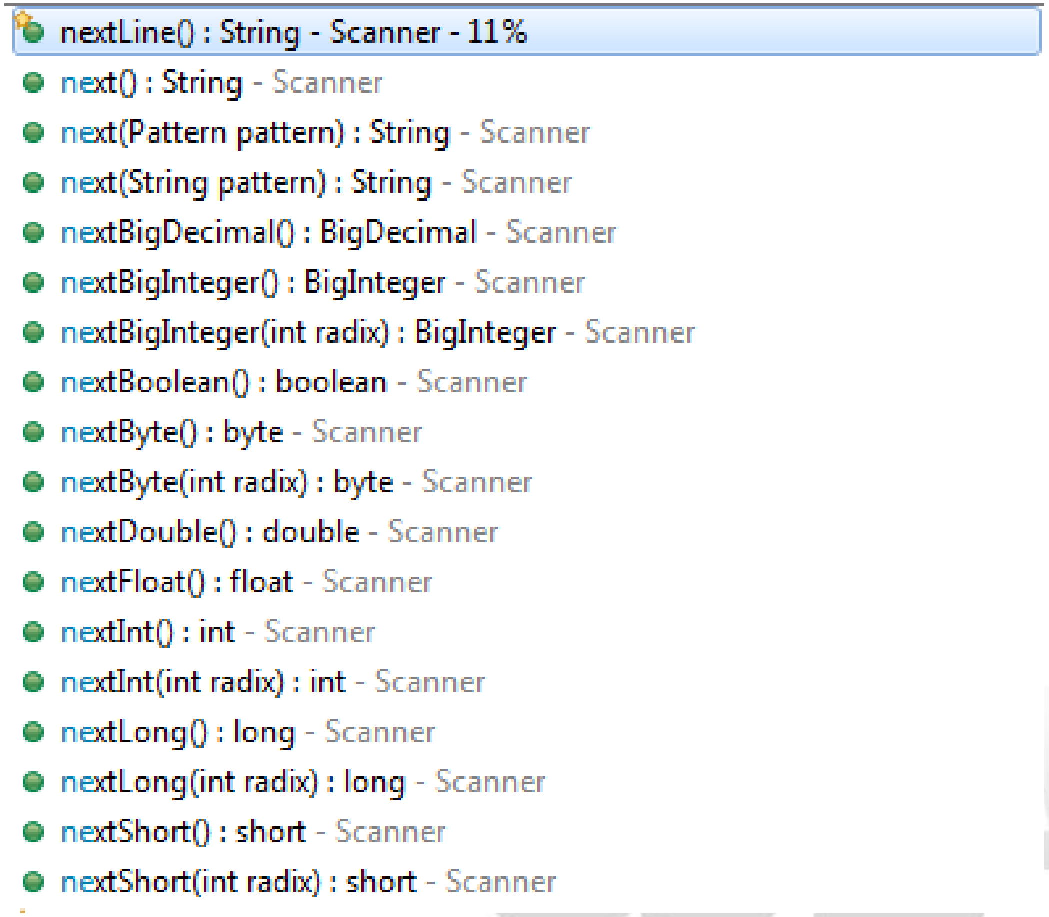
Console Input (Contd)

- To use the Scanner within you applications you must import the Scanner class as follows
  - import java.util.Scanner;

- A Scanner object can be created by declaring a variable as type Scanner
  - Scanner keyboard = new Scanner(System.in);

Console Input (Contd)

- The Scanner object provides us with various methods for reading input.



```
nextLine() : String - Scanner - 11%
next() : String - Scanner
next(Pattern pattern) : String - Scanner
next(String pattern) : String - Scanner
nextBigDecimal() : BigDecimal - Scanner
nextBigInteger() : BigInteger - Scanner
nextBigInteger(int radix) : BigInteger - Scanner
nextBoolean() : boolean - Scanner
nextByte() : byte - Scanner
nextByte(int radix) : byte - Scanner
nextDouble() : double - Scanner
nextFloat() : float - Scanner
nextInt() : int - Scanner
nextInt(int radix) : int - Scanner
nextLong() : long - Scanner
nextLong(int radix) : long - Scanner
nextShort() : short - Scanner
nextShort(int radix) : short - Scanner
```

Console Input (Contd)

- int intValue = keyboard.nextInt();
- double dValue = keyboard.nextDouble();

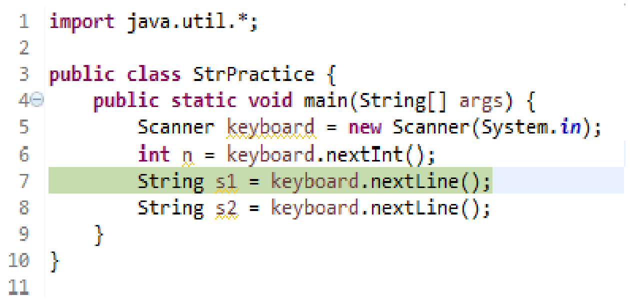
Reads one integer or double value.
Each number separated by a whitespace is considered different.

- Multiple inputs must be separated by whitespace and read by multiple invocations of the appropriate method
  - Whitespace is any string of characters, such as blank spaces, tabs, and line breaks that print out as white space
  - String str = keyboard.next ();

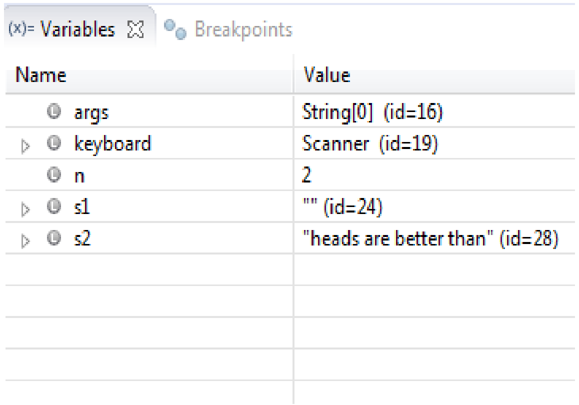
Reads a word/string. – A word is a contiguous string of characters not including whitespaces.

Console Input (Contd)

- **NextLine()**
  - This method reads the remaining strings up to the end of the line and discards the eol (end of line) character.

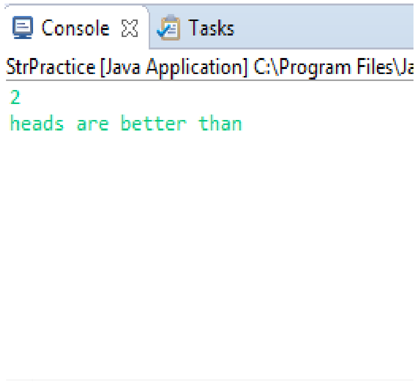


```
import java.util.*;

public class StrPractice {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        int n = keyboard.nextInt();
        String s1 = keyboard.nextLine();
        String s2 = keyboard.nextLine();
    }
}
```



(x)= Variables   Breakpoints

Name	Value
args	String[0] (id=16)
keyboard	Scanner (id=19)
n	2
s1	"" (id=24)
s2	"heads are better than" (id=28)

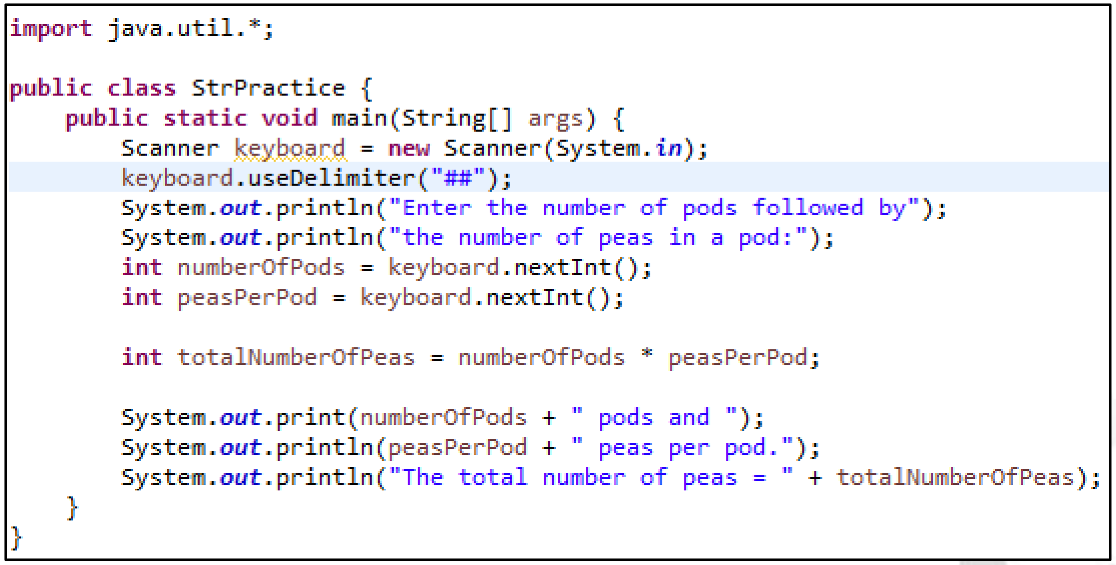


Console   Tasks

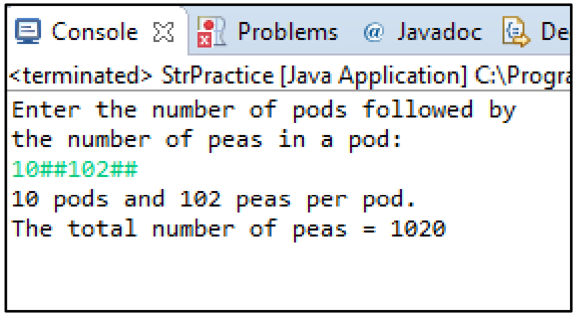
```
StrPractice [Java Application] C:\Program Files\Ja
2
heads are better than
```

Scanner Class

- It is possible to change the delimiters used by the scanner class
- This changes the delimiters from being whitespaces to the user-specified value
  - Scanner keyboard2 = new Scanner(System.in);
  - Keyboard2.useDelimiter("##");



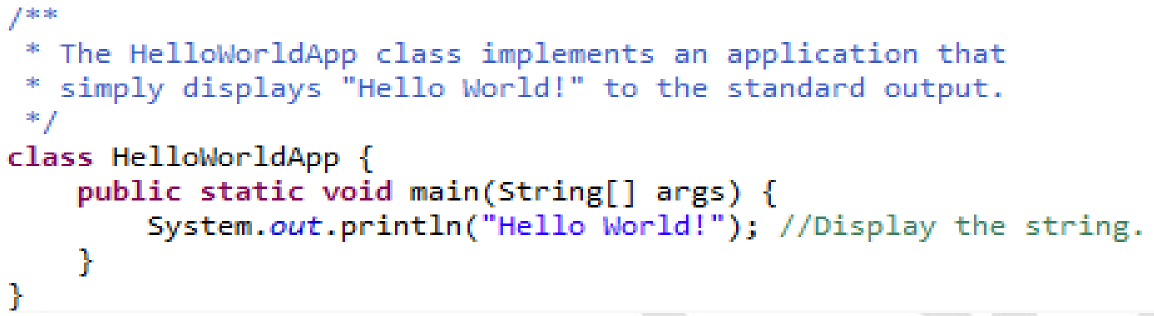
```java
import java.util.*;

public class StrPractice {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        keyboard.useDelimiter("##");
        System.out.println("Enter the number of pods followed by");
        System.out.println("the number of peas in a pod:");
        int numberOfPods = keyboard.nextInt();
        int peasPerPod = keyboard.nextInt();

        int totalNumberOfPeas = numberOfPods * peasPerPod;

        System.out.print(numberOfPods + " pods and ");
        System.out.println(peasPerPod + " peas per pod.");
        System.out.println("The total number of peas = " + totalNumberOfPeas);
    }
}
```



```
<terminated> StrPractice [Java Application] C:\Progra
Enter the number of pods followed by
the number of peas in a pod:
10##102##
10 pods and 102 peas per pod.
The total number of peas = 1020
```

Comments

- Double slashes (//) are used in the Java programming language, and tell the compiler to treat everything from the slashes to the end of the line as text.
- Instead of double slashes, you can use C-style comments (/* */) to enclose one or more lines of code to be treated as text.
- Comments are ignored by the compiler but are useful to other programmers



```
/**
 * The HelloWorldApp class implements an application that
 * simply displays "Hello World!" to the standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); //Display the string.
    }
}
```

Self-Test (2)

- String type 변수인 name을 선언한 뒤, 자신의 영어 이름으로 값을 초기화 할 것(영어 이름은 소문자로)
- String type 변수인 greeting1과 greeting2를 각각 “Hello”와 “nice to meet you”로 초기화 할 것
- String type 변수 uName을 선언한 뒤, 대문자로만 구성된 자신의 영어 이름으로 값을 초기화 할 것(이 때 직접 적지 않고 name 변수와 String class의 method를 활용할 것)
- greeting1, name, greeting2 순으로 3개의 변수를 concatenation하여 출력할 것
- name의 길이를 String class의 method를 활용하여 출력할 것
- String class의 method를 활용하여 name과 uName이 같은지 출력할 것
- String class의 method를 활용하여 name과 uName이 대소문자를 무시했을 때 같은지 출력할 것