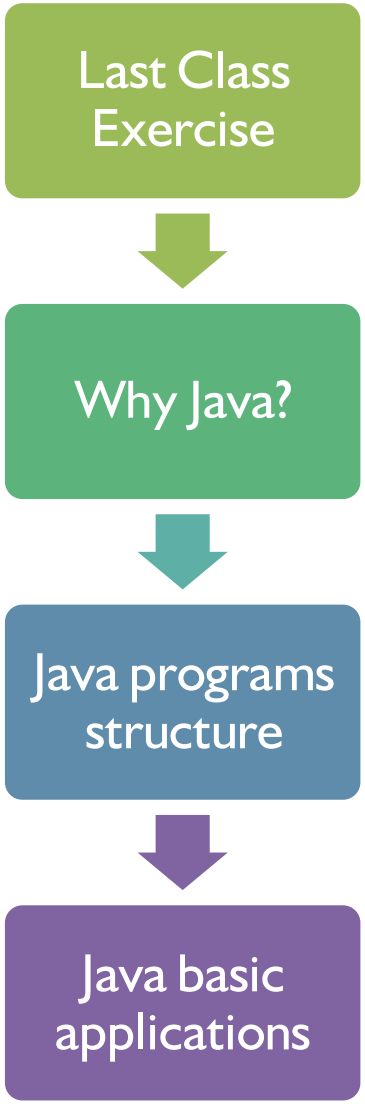
Java Basics I

Christian Rodríguez Bustos

Object Oriented Programming

Agenda


Last Class Exercise
Why Java?
Java programs structure
Java basic applications

Last Class Exercise

1. Do the Eclipse HelloWord!! or NetBeans HelloWord!!
2. Modify the "Multidimensional array use example "code in order to:
   - print the main diagonal of the next two multidimensional arrays

1	2	3
4	5	6
7	8	9

Numbers array

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

Letters array

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

    int array1[][] = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}};

    char array2[][] = {
        {'a', 'b', 'c', 'd'},
        {'e', 'f', 'g', 'h'},
        {'i', 'j', 'k', 'l'},
        {'m', 'n', 'o', 'p'}};

    System.out.println("Values in int array main diagonal are: ");
    outputIntArray(array1);

    System.out.println("Values in char array main diagonal are: ");
    outputCharArray(array2);
}
// ...
```

Int Array

Char Array

```java
// ...
private static void outputIntArray(int[][] array) {
    for (int row = 0; row < array.length; row++) {
        for (int col = 0; col < array.length; col++) {
            if (row == col) {
                System.out.print(array[row][col]);
            } else {
                System.out.print(" ");
            }
        }
        System.out.println("");
    }
}

private static void outputCharArray(char[][] array) {
    for (int row = 0; row < array.length; row++) {
        for (int col = 0; col < array.length; col++) {
            if (row == col) {
                System.out.print(array[row][col]);
            } else {
                System.out.print(" ");
            }
        }
        System.out.println("");
    }
}
```

Method for printing int arrays

Method for printing char arrays

Why Java?

Java advantages

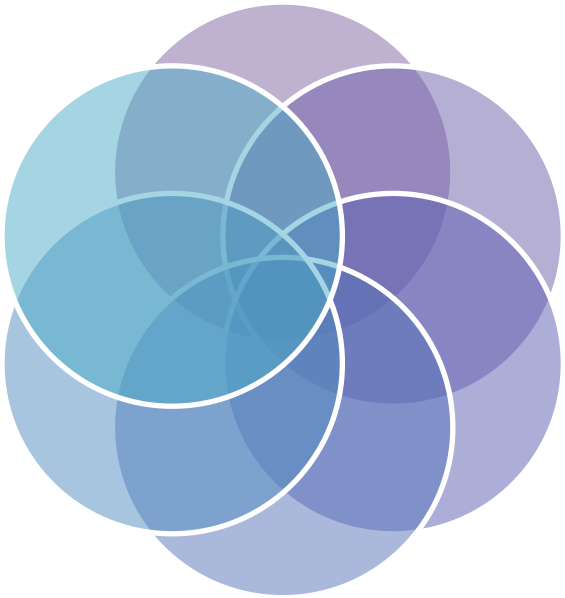
Java acronyms

Java advantages

Java Is Architecture Neutral

Java Is Free!

Java Provides "One-Stop Shopping"



Java Is an Open Standard

Java Is Object-Oriented from the Ground Up

Practice Makes Perfect

Java Is Architecture Neutral

Platform independent Java source code

Java compiler for Windows

Java compiler for Linux

Java compiler for Solaris

```
Êþº¾ □ - □□          □
 □ □□ □ □□ □()V□ □
([Ljava/lang/String;)V□
□<init>□ □Code□ □Hello
□ Hello.java□ ...
```

Platform independent **bytecode**
(.class file)

Platform independent C++ source code

C++ compiler for Windows

C++ compiler for Linux

C++ compiler for Solaris

```
10010011
00100110
10110010
11011001
11001011
```

Windows version

```
10010011
00100110
10110010
11011001
11001011
```

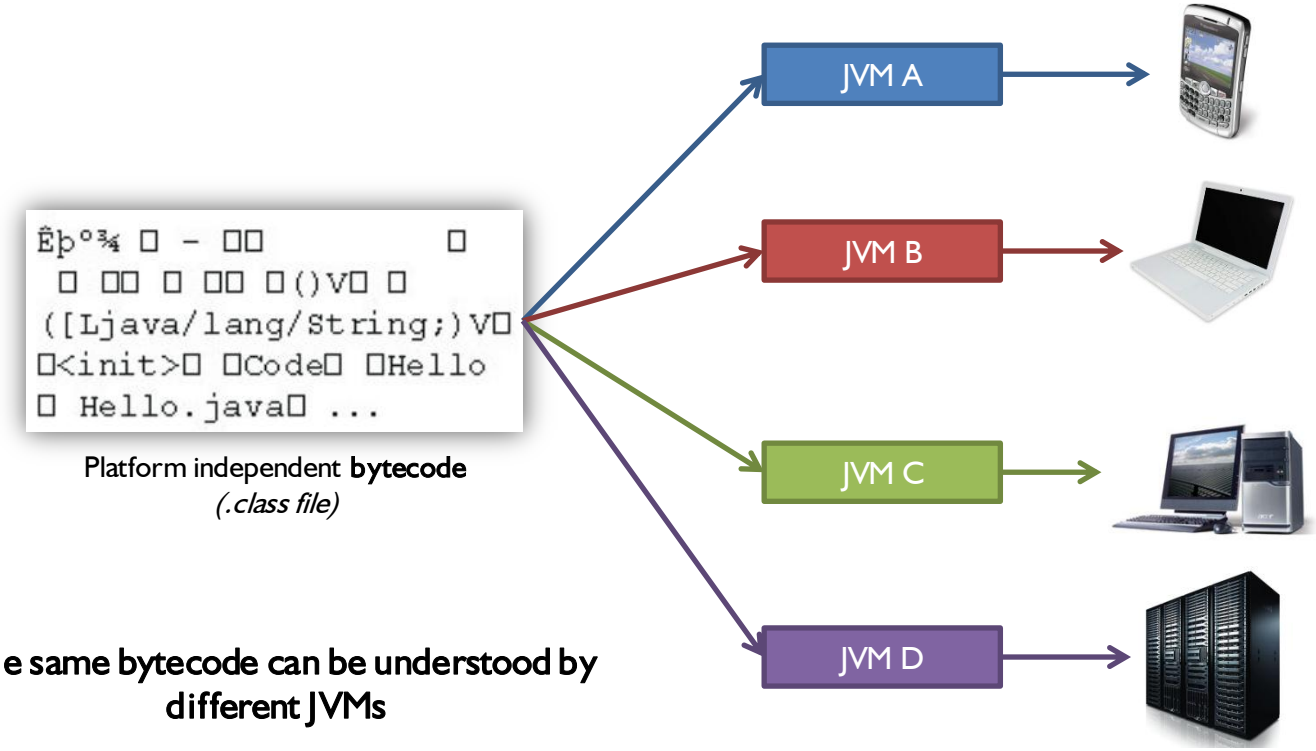
Linux version

```
10010011
00100110
10110010
11011001
11001011
```

Solaris version

Java Is Architecture Neutral

*The **Java Virtual Machine** (JVM)*
converts the compiled Java byte code to machine code.



Platform independent **bytecode**
(.class file)

The same bytecode can be understood by different JVMs

In theory, bytecode is forward compatible with newer versions of the JVM

Java Provides "One-Stop Shopping"

Java language provides an extensive set of **application programming interfaces (APIs)**

java.io: Used for file system access

java.sql: The JDBC API, used for communicating with relational databases in a vendor-independent fashion

java.awt: The Abstract Windowing Toolkit, used for GUI development

javax.swing: Swing components, also used for GUI development

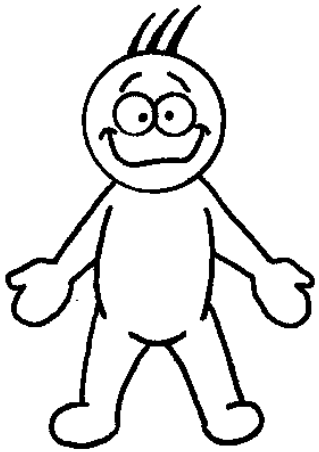
And there are **many** more …

Java Is Object-Oriented from the Ground Up

Primitive or simple data types are still just single pieces of information

Object-oriented objects are complex types that have multiple pieces of information and specific **properties** (or attributes) and **behaviors** (methods).

```
4   public class Person {
5
6       private double height;       // property (atribute)
7       private double weight;       // property (atribute)
8       private int age;             // property (atribute)
9
10      public void walk(int distance){
11          // walk behavior (method)
12      }
13
14      public void sleep(int minutes){
15          // sleep behavior (method)
16      }
17
18  }
```



Java Is Object-Oriented from the Ground Up

All data, with the exception of a few primitive types are **objects**.

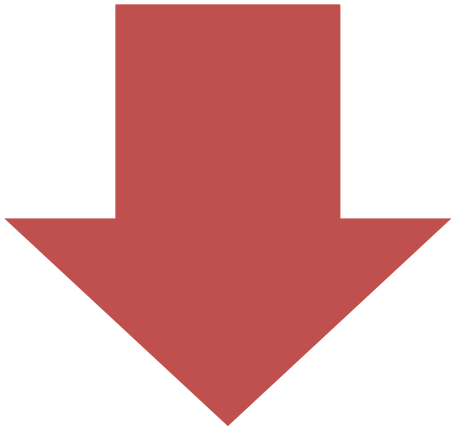
All of the GUI building blocks windows, buttons, text input fields, scroll bars, lists, menus, and so on **are objects**.

All functions are associated with objects and are known as methods there can be no "free-floating" functions as there are in C/C++.

Practice Makes Perfect

Java taken the best features of C++,Eiffel, Ada, and Smalltalk

Added some capabilities and features not found in those languages.



Features that had proven to be most troublesome in those earlier languages were eliminated.

Java Is an Open Standard

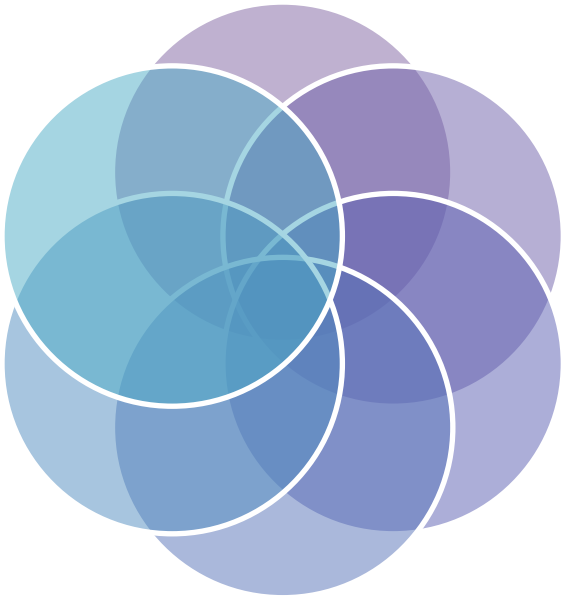
Java Is Free!

Java advantages

Java Is Architecture Neutral

Java Is Free!

Java Provides "One-Stop Shopping"

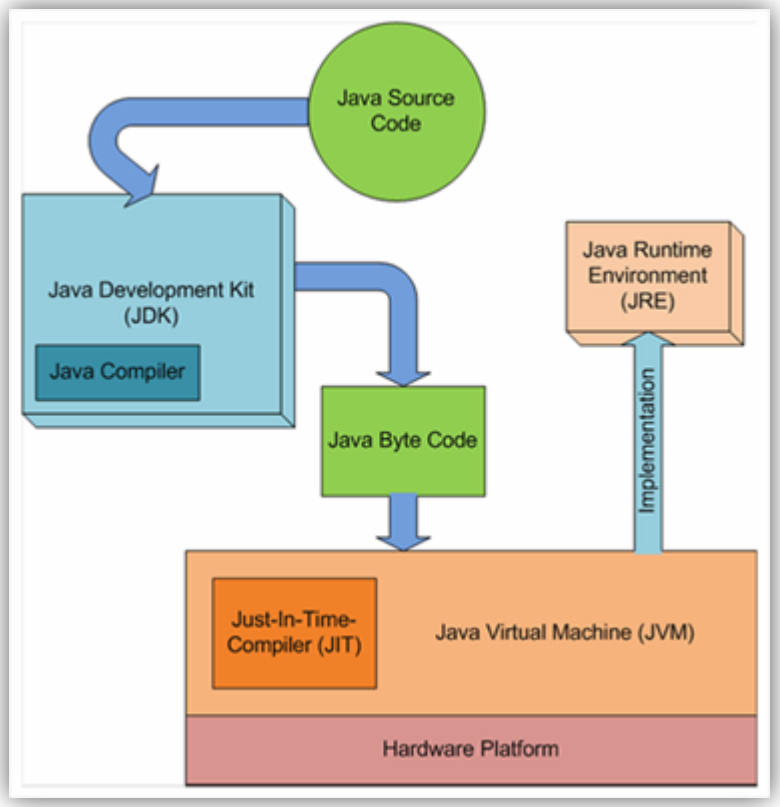


Java Is an Open Standard

Java Is Object-Oriented from the Ground Up

Practice Makes Perfect

Java acronyms



JDK: Java Developer Kit: Develop and execution

JRE: Java Runtime Environment: Execution

JVM: Java Virtual Machine

Java acronyms

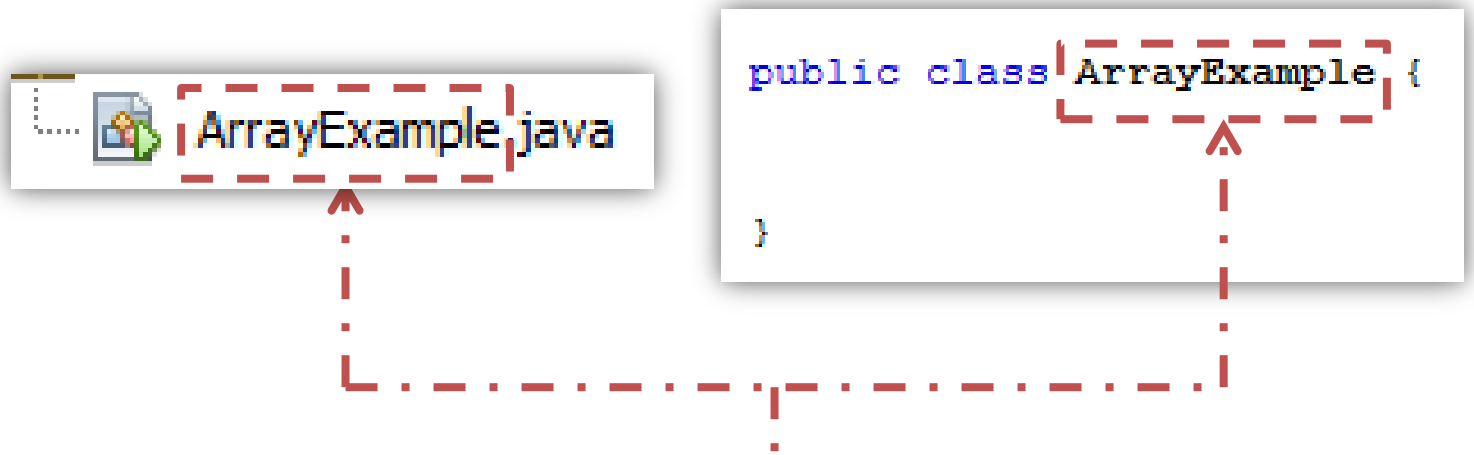
Java programs structure

File Structure

Classes

Methods

A source code file holds **one** class definition



Same name

Put a class in a source file !!!

A class holds **one or more** methods

Method 1

Method 2

Method 3

```
public class ArrayExample {
    public static void main(String[] args) {...}
    private static void outputIntArray(int[][] array) {...}
    private static void outputCharArray(char[][] array) {...}
}
```

Put methods in a class !!!

A method holds **statements**

Method 1
Statements

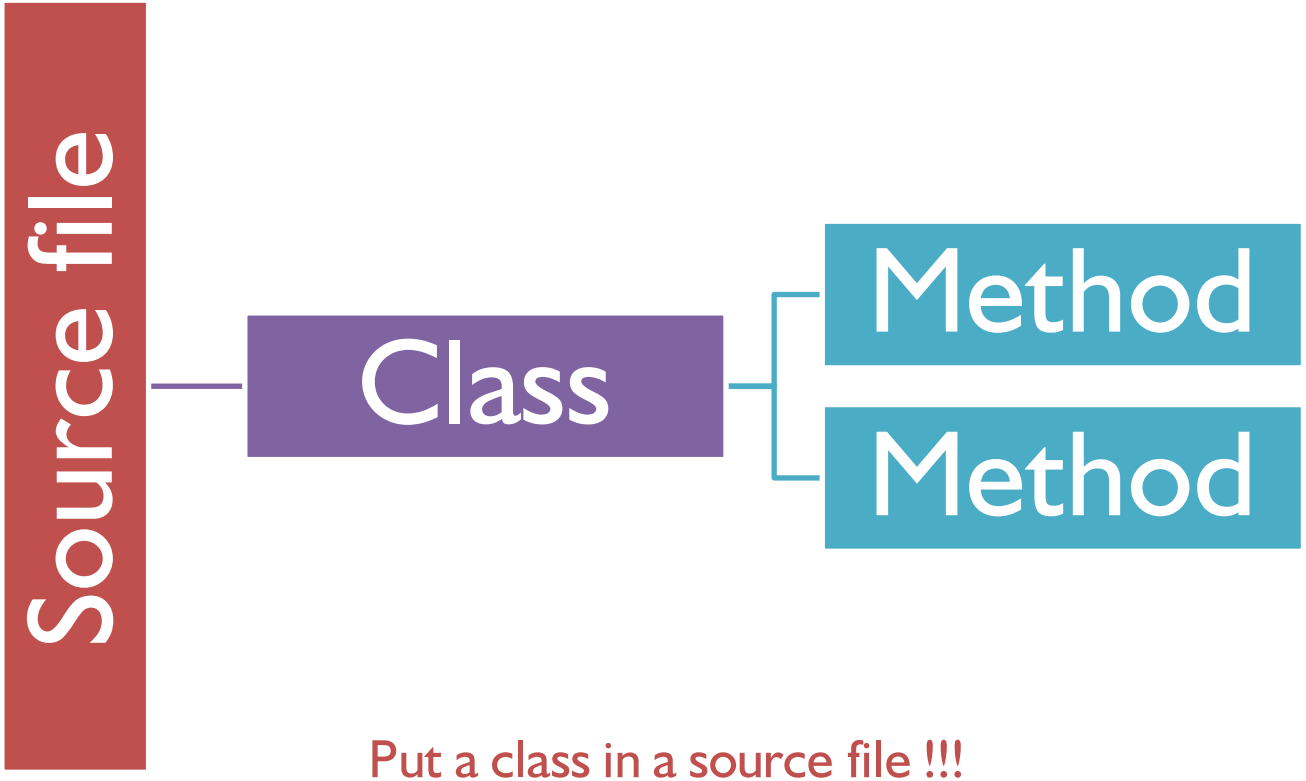
Method 2
Statements

```java
private static void outputIntArray(int[][] array) {
    for (int row = 0; row < array.length; row++) {
        for (int col = 0; col < array.length; col++) {
            if (row == col) {
                System.out.print(array[row][col]);
            } else {
                System.out.print(" ");
            }
        }
        System.out.println("");
    }
}

private static void outputCharArray(char[][] array) {
    for (int row = 0; row < array.length; row++) {
        for (int col = 0; col < array.length; col++) {
            if (row == col) {
                System.out.print(array[row][col]);
            } else {
                System.out.print(" ");
            }
        }
        System.out.println("");
    }
}
```

Put statements in a method !!!

File structure



Put a class in a source file !!!

Put methods in a class !!!

Put statements in a method !!!

Class definition

Define who can access the class:

`public` means everyone can access it

This is a `class`

Class name

```
public class MyFirstClass {

}
```

Closing brace

Opening brace

Class names Should be **nouns**

- Should be **nouns**, in mixed case with the first letter of each internal word capitalized.
- Try to keep your class names **simple and descriptive**.
- Use **whole words**, avoid **acronyms and abbreviations**.
- Java is case sensitive.

Good Examples:

- `class SoccerPlayer {…}`
- `class Person {…}`

Bad Examples

- `class XYZ {…}`
- `class PERSON {…}`
- `class soccerplayer {…}`

Methods definition

Define who can access the class:

`public` means everyone can access it

We will see this later ☺

Method name

Parameters to the method

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

Return type

Closing brace

Opening brace

Methods names Should be **verbs**

Good Examples:

- `private static void play(int coinValue) {…}`
- `public static void moveToRight(int steps) {…}`
- `public static void getDirection() {…}`

Bad Examples

- `public static void person() {…}`
- `public static void PLAY() {…}`
- `public static void soccerplayer() {…}`

Should be **verbs** (behaviors), in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.

The `main` method is where your program start to run

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

}
```

Is not necessary a main method in a class

Java basic applications

Comments, operators and precedence

System.out and System.in

Comments improve readability of source code

Most of the times ;)

```
// drunk, fix later
```

```
//When I wrote this, only God and I understood what I was doing
//Now, God only knows
```

```
// Magic. Do not touch.
```

A good source code do not required comments

```
public static void main(String[] args) {
    /* This is a
     * multiline
     * comment
     */
}

public static void main(int[] args) {
    // This is single line comment
}
```

What is the best comment in source code you have ever encountered

Self explanatory code vs Commented code

Self explanatory

```
public static int calculateRectangleArea(int height, int width) {
    return height * width;
}
```

```
/**
 * This method calculate the area of a rectangle
 * @param a is the height
 * @param b is the width
 * @return the area of a rectangle
 */
public static int method(int a, int b) {
    return a * b;
}
```

Commented code

Precedence of arithmetic operators

Operators	Associativity	Type
* / %	left to right	multiplicative
+ -	left to right	additive
< <= > >=	left to right	relational
== !=	left to right	equality
=	right to left	assignment

Precedence of arithmetic operators

```
public static void main(String[] args) {
    int testPrecedence = 2 * 5 * 5 + 3 * 5 + 7 / (5 + 1 % 2);
    System.out.println("test: " + testPrecedence);

    testPrecedence = 2 * 5 * (5 + 3) * 5 + 7 / 5 + 1 % 2;
    System.out.println("test: " + testPrecedence);

    testPrecedence = 2 * 5 * 5 + 3 * (5 + 7) / (5 + 1) % 2;
    System.out.println("test: " + testPrecedence);
}
```

??????

Precedence of arithmetic operators

```java
public static void main(String[] args) {
    int testPrecedence = 2 * 5 * 5 + 3 * 5 + 7 / (5 + 1 % 2);
    System.out.println("test: " + testPrecedence);

    testPrecedence = 2 * 5 * (5 + 3) * 5 + 7 / 5 + 1 % 2;
    System.out.println("test: " + testPrecedence);

    testPrecedence = 2 * 5 * 5 + 3 * (5 + 7) / (5 + 1) % 2;
    System.out.println("test: " + testPrecedence);
}
```

Output - Assignment00 (run)

```
run:
test: 66
test: 402
test: 50
BUILD SUCCESSFUL (total time: 0 seconds)
```

`System.out` is the "standard" java output stream

This stream is already open and ready to accept output data.

Print an object

```
System.out.print(Object object);
```

Print an object using a specific format

```
System.out.printf(String format, Object object);
```

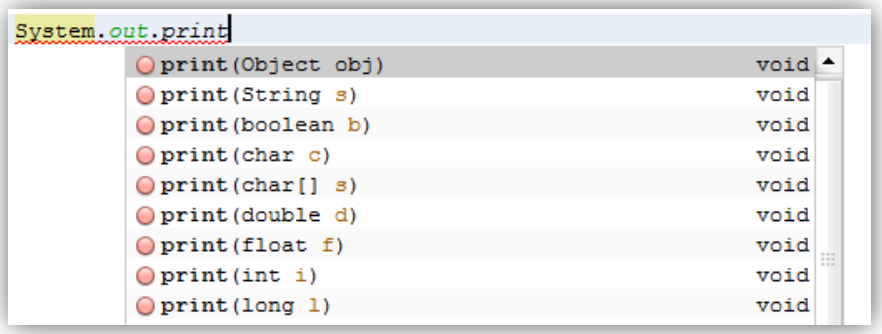
Print an object in a new line

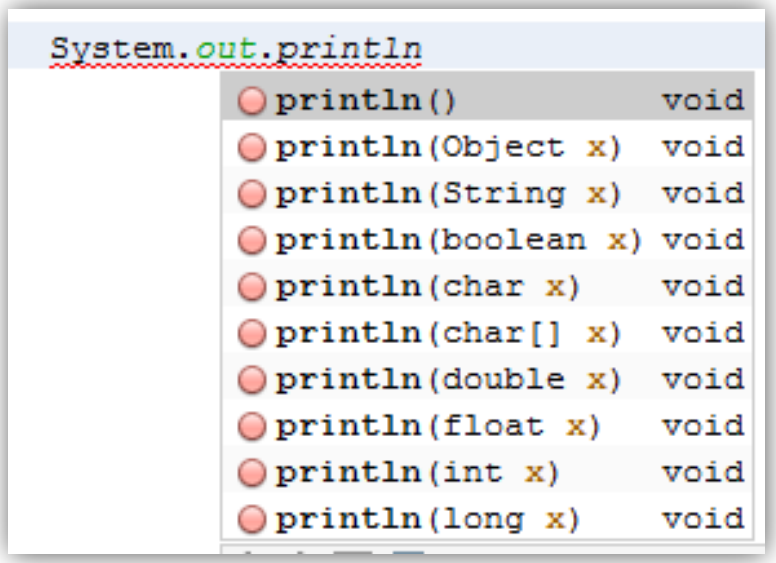
```
System.out.println(Object object);
```

System.out is the "standard" java output stream

```
System.out.print();
```

Print without moving cursor to the next line



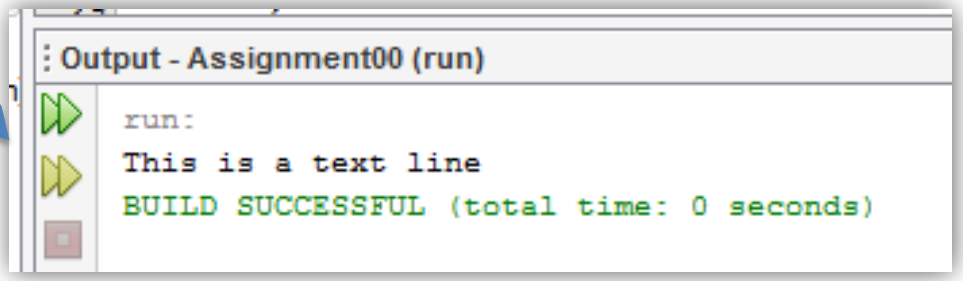



```
System.out.println();
```

Print moving cursor to the next line

System.out is the "standard" java output stream

```
System.out.print("This is a ");
System.out.println("text line");
```



System.out is the "standard" java output stream

```
System.out.printf();
```

Print without moving cursor to the next line

```
System.out.printf
        printf(String format, Object... args) PrintStream
        printf(Locale l, String format, Ob... PrintStream
```

```
Output - Assignment00 (run)
run:
This is a text line
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
System.out.printf("%s","This is a text line");
System.out.println();
```

Printing formats: Numbers with zeros

```java
public class FormatOutputExample {

    public static void main(String[] args) {

        System.out.printf("Num is %03d\n", 5);

    }
}
```

```
Output - OOP20112 (run)
run:
Num is 005
BUILD SUCCESSFUL (total time: 3 seconds)
```

```java
"Num is %03d\n"
```

This is the format

Printing formats: Letters

```java
public class FormatOutputExample {

    public static void main(String[] args) {

        System.out.printf("Char values is %c\n", 'c');

    }
}
```

```java
"Char values is %c\n"
```

This is the **format**

```
Output - OOP20112 (run)
run:
Char values is c
BUILD SUCCESSFUL (total time: 0 seconds)
```

Printing formats: Dates

Remember to import the **Date Class**

```
import java.util.Date;

public class FormatOutputExample {

    public static void main(String[] args) {

        Date date = new Date();
        System.out.printf("The date is %s\n", date);
    }
}
```

```
"The date is %s\n"
```

This is the **format**

```
Output - OOP20112 (run)
run:
The date is Mon Aug 08 18:24:01 COT 2011
BUILD SUCCESSFUL (total time: 2 seconds)
```

Printing formats: Floats

```java
public class FormatOutputExample {

    public static void main(String[] args) {

        System.out.printf("The first two PI decimals are: %.2f\n", Math.PI);
    }
}
```

Output - OOP20112 (run)

```
run:
The first two PI digits are: 3.14
BUILD SUCCESSFUL (total time: 1 second)
```

```java
"The first two PI decimals are: %.2f\n"
```

This is the **format**

Printing formats resources

See more formats on

http://www.java2s.com/Code/JavaAPI/java.lang/System.out.printf.htm

Escape sequences

Escape sequence	Description
`\n`	Newline. Position the screen cursor at the beginning of the next line.
`\t`	Horizontal tab. Move the screen cursor to the next tab stop.
`\r`	Carriage return. Position the screen cursor at the beginning of the current line—do not advance to the next line. Any characters output after the carriage return overwrite the characters previously output on that line.
`\\`	Backslash. Used to print a backslash character.
`\"`	Double quote. Used to print a double-quote character. For example, `System.out.println( "\"in quotes\"" );` displays `"in quotes"`

Escape sequences: new line and tab examples

```
System.out.print("Line with a new line sequence escape \n");
```

Output - OOP20112 (run)

```
run:
Line with a new line sequence escape
BUILD SUCCESSFUL (total time: 1 second)
```

```
System.out.print("Line with a \t tabulation sequence escape \n");
```

Output - OOP20112 (run)

```
run:
Line with a      tabulation sequence escape
BUILD SUCCESSFUL (total time: 1 second)
```

Escape sequences: slash and double quote examples

```
System.out.print("Line with a \\ slash sequence escape \n");
```

Output - OOP20112 (run)

```
run:
Line with a \ slash sequence escape
BUILD SUCCESSFUL (total time: 1 second)
```

```
System.out.print("Line with a \" double quote sequence escape ");
```

Output - OOP20112 (run)

```
run:
Line with a " double quote sequence escape
BUILD SUCCESSFUL (total time: 1 second)
```

System.in is used to read user inputs

System.in and **Scanner** class allow us to read values typed by the user

```
import java.util.Scanner;

public class UserInputReaderExample {
    // ...
```

First we need to import the **Scanner class** at the beginning of our source code file

System.in: reading strings example

Creating the scanner

Reading an integer

```
3   import java.util.Scanner;
4
5   public class UserInputReaderExample {
6
7       public static void main(String[] args) {
8
9           Scanner reader = new Scanner(System.in);
10
11          int age;
12
13          System.out.print("Please enter your age: ");
14          age = reader.nextInt();
15
16          System.out.println("Your age is " + age);
17      }
18  }
```

Output - OOP20112 (run)

```
run:
Please enter your age: 1000
Your age is 1000
BUILD SUCCESSFUL (total time: 6 seconds)
```

System.in: reading strings example

Creating the scanner

Reading a String

```
3   import java.util.Scanner;
4
5   public class UserInputReaderExample {
6
7       public static void main(String[] args) {
8
9           Scanner reader = new Scanner(System.in);
10
11          String name;
12
13          System.out.print("Please enter your name: ");
14          name = reader.nextLine();
15
16          System.out.println("Your name is " + name);
17      }
18  }
```

Output - OOP20112 (run)

```
run:
Please enter your name: OOP Man
Your name is OOP Man
BUILD SUCCESSFUL (total time: 13 seconds)
```

System.in: reading posibilities

```
nextBigDecimal()            BigDecimal
nextBigInteger()            BigInteger
nextBigInteger(int radix)   BigInteger
nextBoolean()               boolean
nextByte()                  byte
nextByte(int radix)         byte
nextDouble()                double
nextFloat()                 float
nextInt()                   int
nextInt(int radix)          int
nextLong()                  long
nextLong(int radix)         long
nextShort()                 short
nextShort(int radix)        short
```

References

- [Barker] J. Barker, *Beginning Java Objects: From Concepts To Code*, Second Edition, Apress, 2005.
- [Deitel] H.M. Deitel and P.J. Deitel, *Java How to Program: Early Objects Version*, Prentice Hall, 2009.
- [Sierra] K. Sierra and B. Bates, *Head First Java*, 2nd Edition, O'Reilly Media, 2005.
- **Code Conventions for the Java Programming Language**, available at http://java.sun.com/docs/codeconv/CodeConventions.pdf