Java Programming Language

Tin Huynh, Ph.D

Contacts

Email & FB: tin.huynh.uit@gmail.com

Skype: tin-huynh

Objectives

- Nắm vững các khái niệm, đặc điểm cơ bản của ngôn ngữ lập trình java.
- Hiểu rõ tư tưởng hướng đối tượng của java
- Có khả năng đọc hiểu tài liệu tiếng Anh về java.
- Có khả năng vận dụng các kỹ thuật lập trình cơ bản và nâng cao trong Java để xây dựng chương trình ứng dụng.
- Có khả năng làm việc nhóm trong lập trình

Objectives

- Hình thành được tư duy tổ chức kiến trúc chương trình một cách có hệ thống và tác phong lập trình chuyên nghiệp thông qua ngôn ngữ Java.
- Có khả năng tự học và sử dụng thành thạo một số công cụ phổ biến hỗ trợ lập trình java

Contents

Assessment

- Đồ án: 50%
- Thực hành: 30%
- Seminar/Bài tập: 20%

Some Popular References

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html

https://www.w3schools.com/java/default.asp

https://www.tutorialspoint.com/java/index.htm

What is Java?

- **Programming language** and a **platform**
- A high level, robust, secured and object-oriented programming language.
- **Platform**: **Any hardware or software environment** in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.

An Example

```
import java.io.*;

class HelloWorld {

        public static void main(String args[]) {

                System.out.println("Hello Class");

        }

}
```

Where it is used?

- According to Sun, 3 billion devices run java
  1. **Desktop Applications** such as acrobat reader, media player, antivirus etc.
  2. **Web Applications** such as irctc.co.in, javatpoint.com etc.
  3. **Enterprise Applications** such as banking applications.
  4. **Mobile**
  5. **Embedded System**
  6. **Smart Card**
  7. **Robotics**
  8. **Games** etc.

Types of Java Applications

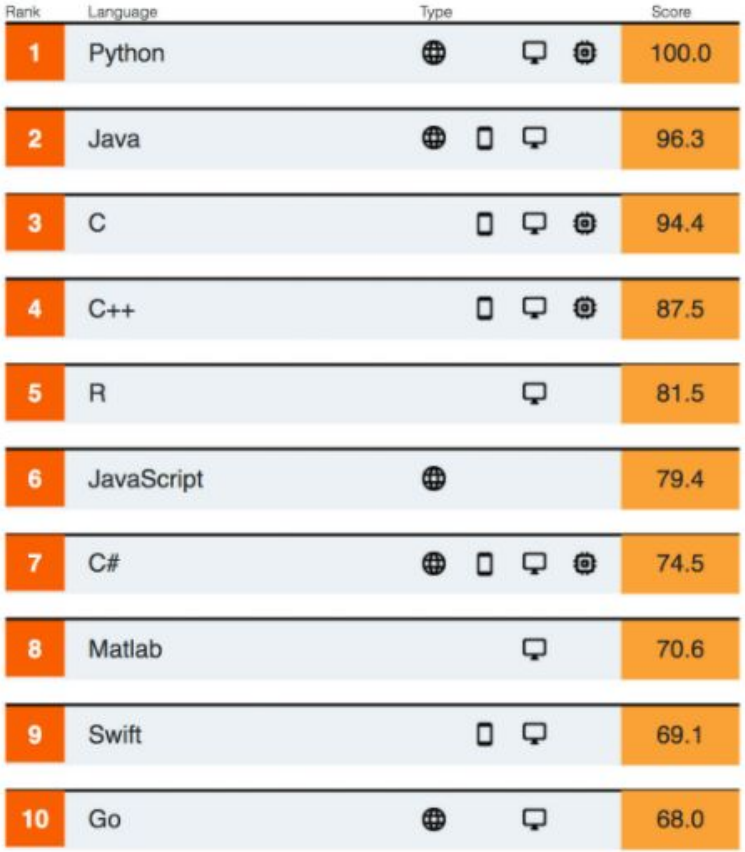
1) Standalone & Desktop Application

2) Web Application & Enterprise Application

3) Mobile Application

The top programming languages

(https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019)

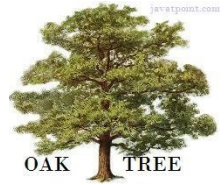


Rank	Language	Type	Score
1	Python		100.0
2	Java		96.3
3	C		94.4
4	C++		87.5
5	R		81.5
6	JavaScript		79.4
7	C#		74.5
8	Matlab		70.6
9	Swift		69.1
10	Go		68.0

History of Java

https://vi.wikipedia.org/wiki/James_Gosling

History of Java



- **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** (**Green Team**), June, 1991
- Develop a language for small, **embedded systems** in electronic appliances like set-top boxes.
- Firstly, it was called **"Greentalk"** by James Gosling and file extension was .gt.
- After that, it was called **Oak**. (symbol of strength and chosen as a national tree of many countries like U.S.A., France, Germany)
- In 1995, Oak was renamed as **"Java“**

History of Java

- The team wanted something that reflected the essence of the technology: revolutionary, dynamic, lively, cool, unique, and easy to spell and fun to say.
- According to James Gosling "Java was one of the top choices along with **Silk**". Since java was so unique, most of the team members preferred java.
- Java is an island of Indonesia where first coffee was produced (called java coffee).
- In 1995, Time magazine called **Java one of the Ten Best Products of 1995**.
- JDK 1.0 released in(January 23, 1996).

History of Java

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan, 1996)
- JDK 1.1 (19th Feb, 1997)
- J2SE 1.2 (8th Dec, 1998)
- J2SE 1.3 (8th May, 2000)
- J2SE 1.4 (6th Feb, 2002)
- J2SE 5.0 (30th Sep, 2004)
- Java SE 6 (11th Dec, 2006)
- Java SE 7 (28th July, 2011)
- Java SE 8 (18th March, 2014)
- Java 9 (Sep 2017)
- Java 10 (Mar 2018)
- Java 11 (Sep 2018)
- Java 12 (Mar 2019)
- Java 13 (Sep 2019)
- Java 14 (Mar 2020)

Features of Java

- Simple
- Object-Oriented
- Platform independent
- Secured
- Robust
- Architecture neutral
- Portable
- Dynamic
- Interpreted
- High Performance
- Multithreaded
- Distributed

Simple?

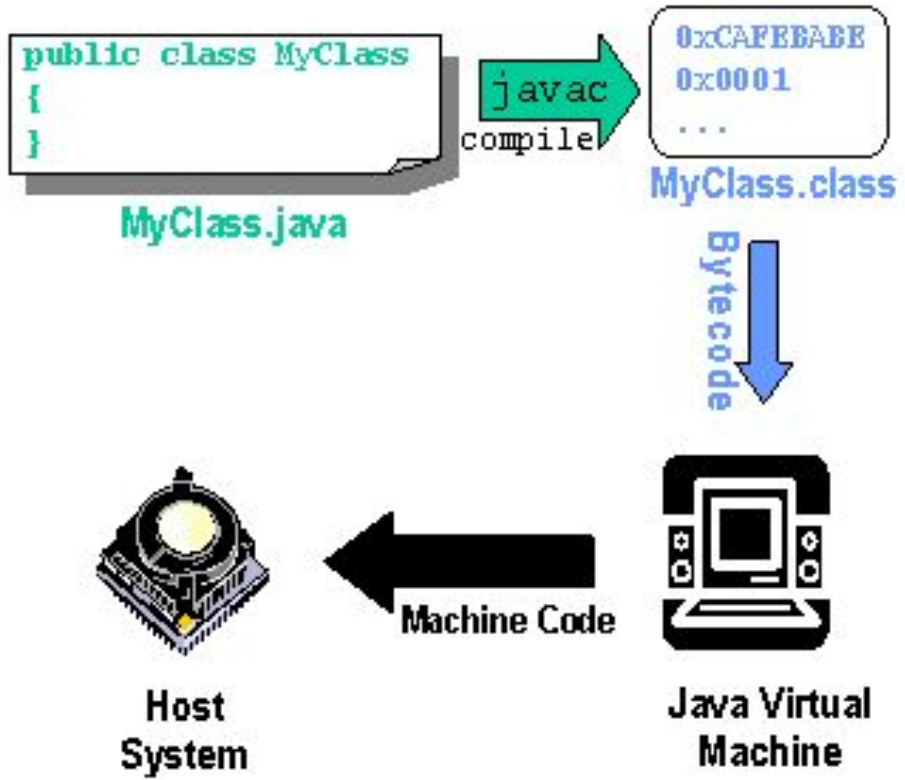
- **Syntax is based on C++** (so easier for programmers to learn it after C++).
- **Removed many confusing rarely-used features**, e.g., explicit pointers, operator overloading etc.
- No need to remove unreferenced objects because there is **Automatic Garbage Collection in java**.

Object oriented?

- Object-oriented means we **organize our software as a combination of different types of objects**.
- **OOPs** is a methodology that **simplify software development and maintenance.**

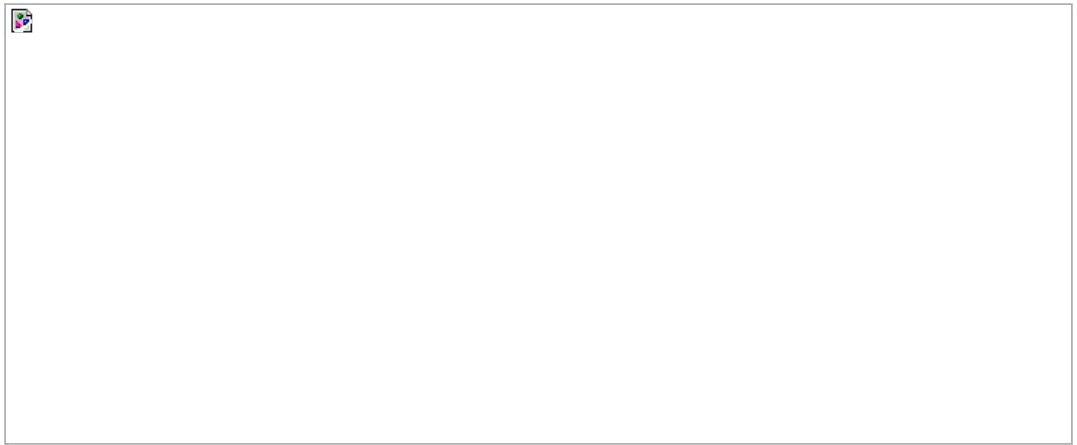
- Basic concepts of OOP:
  - *Object*
  - *Class*
  - *Inheritance*
  - *Polymorphism*
  - *Abstraction*
  - *Encapsulation*

Platform Independent (Write Once & Run Anywhere)



Secured?

- **Authentication techniques are based on public-key encryption**.
- **No explicit pointer**
- **Java Programs run inside virtual machine sandbox.**
- **Security Manager:** determines what resources a class can access such as reading and writing to the local disk.



Robust

- Robust simply means strong. Java uses **strong memory management**. There are **lack of pointers** that avoids security problem. There is **automatic garbage collection** in java. There is **exception handling** and **type checking mechanism** in java. All these points makes java robust.

Architecture - neural (Trung lập kiến trúc)

- There is no implementation dependent features e.g. **size of primitive types is fixed**.
- In C programming, **int data type** occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, **it occupies 4 bytes of memory for both 32 and 64 bit architectures**.

Portable

- We may carry the java bytecode to any platform.

Performance

- Slower than a compiled language (e.g., C++)

Distributed

- RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet.

Multi-threaded

- We can write Java programs that deal with many tasks at once by defining multiple threads.

C++ & Java

	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in window, web-based, enterprise and mobile applications.
Goto	C++ supports goto statement.	Java doesn't support goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	C++ supports operator overloading.	Java doesn't support operator overloading.

C++ & Java

	C++	Java
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. But you can't write the pointer program in java. It means java has restricted pointer support in java.
Compiler and Interpreter	C++ uses compiler only.	Java uses compiler and interpreter both.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for	Java has built-in thread support.

C++ & Java

	C++	Java
Documentation comment	C++ doesn't support documentation comment.	Java supports documentation comment (/** ... */) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.

C++ & Java

	C++	Java
Inheritance Tree	C++ creates a new inheritance tree always.	Java uses single inheritance tree always because all classes are the child of Object class in java. Object class is the root of inheritance tree in java.

Requirement for Hello Java Example

- Install the JDK if you don't have installed it, download the JDK and install it.
- Set path of the jdk/bin directory.

  (Ref: http://www.javatpoint.com/how-to-set-path-in-java)
- Create the java program.
- Compile and run the java program.

How many ways can we write a java program

1. **By changing sequence of the modifiers, method prototype is not changed.**

Let's see the simple code of main method.

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

2. **Subscript notation in java array can be used after type, before variable or after variable.**

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

How many ways can we write a java program

3) You can provide var-args support to main method by passing 3 ellipses (dots)

```
public static void main(String... args)
```

4) Having semicolon at the end of class in java is optional.

```
class A{
    static public void main(String... args){
        System.out.println("....");
    }
};
```

Valid java main method signature

- **public static void** main(String[] args)
- **public static void** main(String []args)
- **public static void** main(String args[])
- **public static void** main(String... args)
- **static public void** main(String[] args)
- **public static final void** main(String[] args)
- **final public static void** main(String[] args)

Invalid java main method signature

- **public void** main(String[] args)
- **static void** main(String[] args)
- **public void static** main(String[] args)
- **abstract public static void** main(String[] args)

JAVA BASIC SYNTAX

Contents

- Identifiers
- Data Types
- Variable
- Modifiers
- Control statements
- Good Programming Style

Java Identifiers

- **Names** used for **classes**, **variables** and **methods** are **called identifiers**.
- Should begin with a letter (**A to Z or a to z**), currency character **($)** or an underscore (_).
- A keyword cannot be used as an identifier.
- Case sensitive.

Java Identifiers

- **age, $salary, _value, __1_value.**

→ **(**legal)

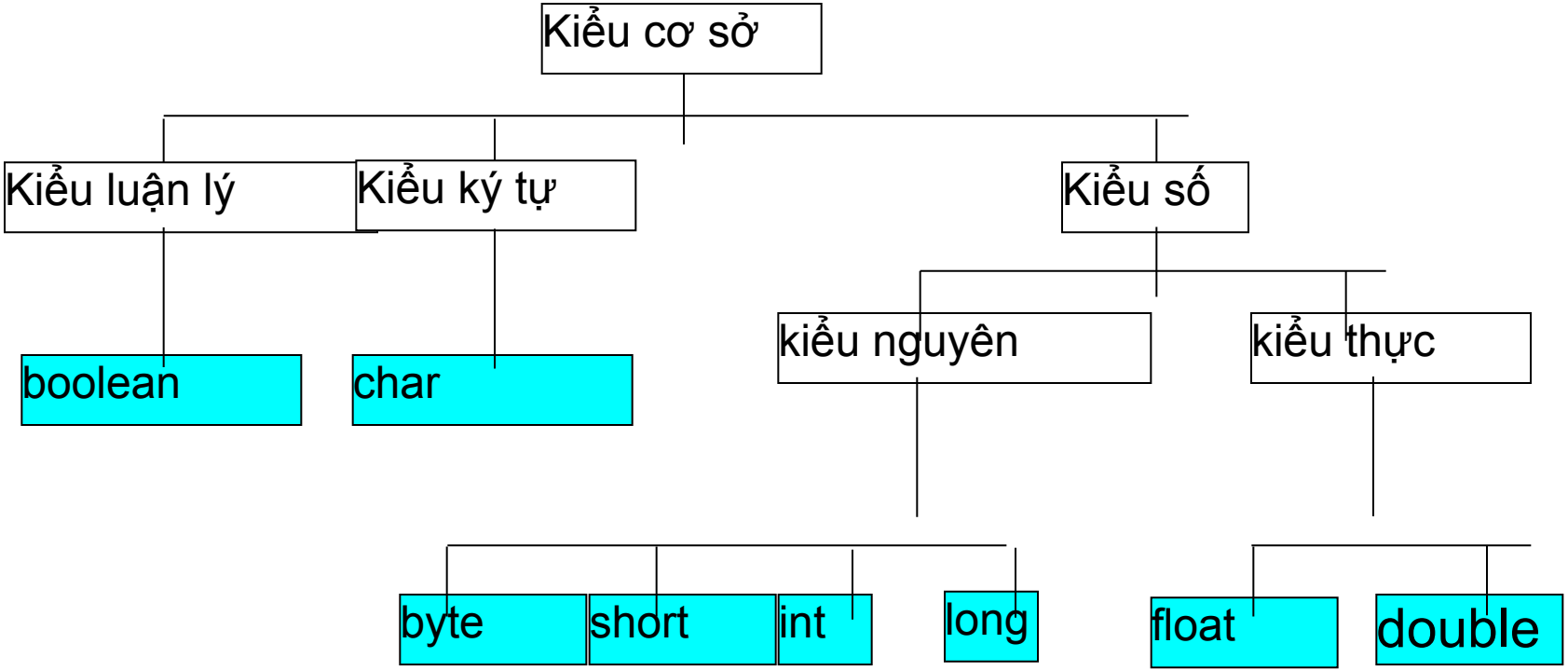
- **123abc, -salary**

→ **(il**legal)

Java Data Types

- Primitive Data Types
- Reference Data Types

Primitive Data Types



Primitive Data Types

Kiểu	Kích thước (bits)	Giá trị	Giá trị mặc định
boolean	[*Note:* The representation of a boolean is specific to the Java Virtual Machine on each computer platform.]	true và false	**false**
char	16	'\u0000' to '\uFFFF' (0 to 65535)	**null**
byte	8	–128 to +127 (-2^7 to $2^7 - 1$)	**0**
short	16	–32,768 to +32,767 (-2^{15} to $2^{15} - 1$)	**0**
int	32	–2,147,483,648 to +2,147,483,647 (-2^{31} to $2^{31} - 1$)	**0**
long	64	–9,223,372,036,854,775,808 to +9,223,372,036,854,775,807 (–263 to 263 – 1)	**0l**
float	32	1.4012984632481707e–45 to 3.4028234663852886E+38	**0.0f**
double	64	4.94065645841246544e–324 to 1.7976931348623157E+308	**0.0d**

Primitive Data Types
(Java Type Casting)

- **Java Type Casting**: Type casting is when you assign a value of one primitive data type to another type.

  ✔ **Narrowing Casting (manually)**: converting a larger type to a smaller size type

  double -> float -> long -> int -> char -> short -> byte

  ✔ **Widening Casting (automatically)**

  byte -> short -> char -> int -> long -> float -> double

Primitive Data Types
(Java Type Casting)

- **Lưu ý**

1. *Không thể chuyển đổi giữa kiểu boolean với int và ngược lại.*
2. ***Nếu** 1 toán hạng kiểu **double** thì*

   *"Toán hạng kia chuyển thành **double**"*

   ***Nếu** 1 toán hạng kiểu **float** thì*

   *"Toán hạng kia chuyển thành **float**"*

   ***Nếu** 1 toán hạng kiểu **long** thì*

   *"Toán hạng kia chuyển thành **long**"*

   ***Ngược lại** "Tất cả chuyển thành **int** để tính toán"*

Primitive Data Types
(Java Type Casting)

- **Example**

1. byte x = 5;

2. byte y = 10;

3. byte z = x + y;

// Dòng lệnh thứ 3 báo lỗi chuyển kiểu cần sửa lại

// byte z = (byte) (x + y);

Reference Data Types

- **Array**

✔Mảng là tập hợp các phần tử có cùng tên và cùng kiểu dữ liệu.

✔Mỗi phần tử được truy xuất thông qua chỉ số

- **Khai báo mảng**

<kiểu dữ liệu>[] <tên mảng>; // mảng 1 chiều

<kiểu dữ liệu> <tên mảng>[]; // mảng 1 chiều

<kiểu dữ liệu>[][] <tên mảng>; // mảng 2 chiều

<kiểu dữ liệu> <tên mảng>[][]; // mảng 2 chiều

Reference Data Types

- **Khởi tạo**

  int arrInt[] = {1, 2, 3};

  char arrChar[] = {'a', 'b', 'c'};

  String arrString[] = {"ABC", "EFG", "GHI"};

- **Cấp phát & truy cập mảng**

  *int [] arrInt = **new** int[100];*

  int arrInt[100]; // Khai báo này trong Java sẽ bị báo lỗi.

  Chỉ số mảng ***n*** phần tử: từ ***0*** đến ***n-1***

Reference Data Types

- **Kiểu đối tượng**

  Khai báo đối tượng

  <Kiểu đối tượng> <biến ĐT>;

  Khởi tạo đối tượng

  *<Kiểu đối tượng> <biến ĐT> = **new** <Kiểu đối tượng>;*

  Truy xuất thành phần đối tượng

  <biến ĐT>.<thuộc tính>

  <biến ĐT>.<phương thức>

Examples

```
int a = 2; // a = 2
double a = 2; // a = 2.0 (Implicit)
int a = 18.7; // ERROR
int a = (int)18.7; // a = 18
double a = 2/3; // a = 0.0
double a = (double)2/3; // a = 0.6666…
```

Examples

- double a = 5.0/2.0; // ?
- int b = 4/2; // ?
- int c = 5/2; // ?
- double d = 5/2; // ?

2.5; 2; 2; 2.0

Examples

```
public static void printSquare(int x) {
    System.out.println(x * x);
}
public static void main(String[] args) {
    int value = 5;
    printSquare(value);
    printSquare(10);
    printSquare(value * 2);
}
```

- 25, 100, 100

Examples

```
public static void printSquare(int x) {
    System.out.println(x * x);
}
public static void main(String[] args) {
    printSquare("hello");
    printSquare(5.5);
}
```

- Errors: Incompatible types

Examples

```
public static void printSquare(double x) {
    System.out.println(x * x);
}
public static void main(String[] args) {
    printSquare(5);
}
```

Examples

```
public static void printSquare(int x) {
    System.out.println("printSquare x = " + x);
    x = x * x;
    System.out.println("printSquare x = " + x);
}

public static void main(String[] arguments) {
    int x = 5;
    System.out.println("main x = " + x);
    printSquare(x);
    System.out.println("main x = " + x);
}
```

- main x = 5; printSquare x = 5;
- printSquare x = 25; main x = 5

Java Variables

Variables are containers for storing data values.

- Local Variables
- Instance Variables
- Class Variables

Local Variables

- **Visible** only within the declared method, constructor or block.
- **An initial value** should be assigned before the first use.
- **Access modifiers cannot** be used for local variables.

tutorialspoint  SIMPLY EASY LEARNING Javac 1.8.x

Execute | Main File

```
1  public class Test{
2     public void uitAge(){
3        int age = 0;
4        age = age + 7;
5        System.out.println("UIT's age is : " + age);
6     }
7
8     public static void main(String args[]){
9        Test test = new Test();
10       test.uitAge();
11    }
12 }
```

Result

Compiling the source code....
$javac Test.java 2>&1

Executing the program....
$java -Xmx128M -Xms16M Test

```
UIT's age is : 7
```

tutorialspoint  SIMPLY EASY LEARNING Javac 1.8.x

Execute | Main File

```
1  public class Test{
2     public void uitAge(){
3        int age;
4        age = age + 11;
5        System.out.println("UIT's age is : " + age);
6     }
7
8     public static void main(String args[]){
9        Test test = new Test();
10       test.uitAge();
11    }
12 }
```

Result

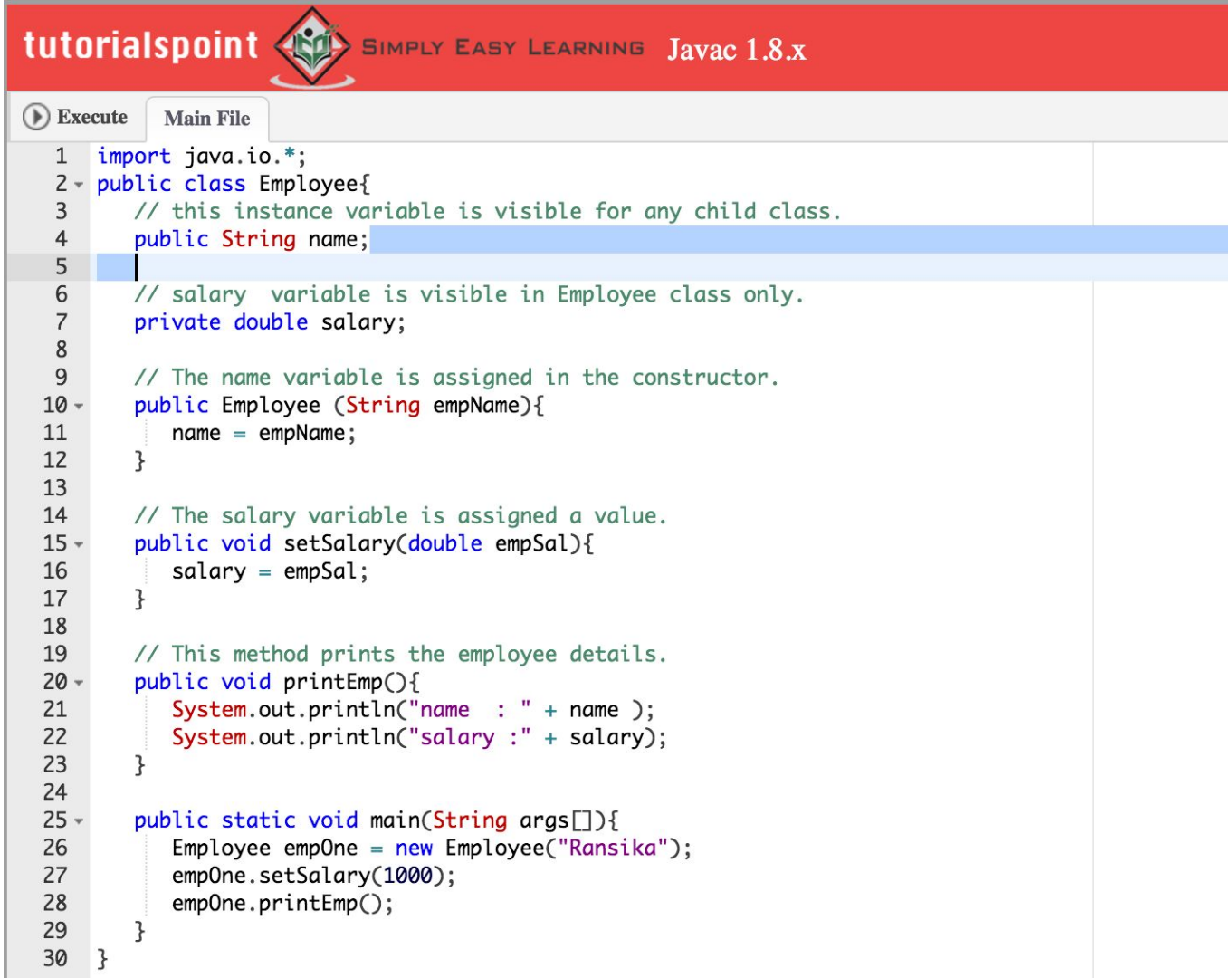
Compiling the source code....

$javac Test.java 2>&1

```
Test.java:4: error: variable age might not have been initialized
      age = age + 11;
            ^
1 error
```

Instance variables

- **Instance variables are declared in a class, but outside a method, constructor or any block.**
- When a space is allocated for an object in the heap, a slot for each instance variable value is created.
- Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed.




```java
import java.io.*;
public class Employee{
   // this instance variable is visible for any child class.
   public String name;

   // salary  variable is visible in Employee class only.
   private double salary;

   // The name variable is assigned in the constructor.
   public Employee (String empName){
      name = empName;
   }

   // The salary variable is assigned a value.
   public void setSalary(double empSal){
      salary = empSal;
   }

   // This method prints the employee details.
   public void printEmp(){
      System.out.println("name  : " + name );
      System.out.println("salary :" + salary);
   }

   public static void main(String args[]){
      Employee empOne = new Employee("Ransika");
      empOne.setSalary(1000);
      empOne.printEmp();
   }
}
```

Instance variables

- Instance variables hold values that must be referenced by more than one method, constructor or block, or essential parts of an object's state that must be present throughout the class.
- Instance variables can be declared in class level before or after use.
- Access modifiers can be given for instance variables
- The instance variables are visible for all methods, constructors and block in the class. Normally, it is recommended to make these variables private (access level). However, visibility for subclasses can be given for these variables with the use of access modifiers.

Instance variables

- Instance variables have default values. For numbers, the default value is 0, for Booleans it is false, and for object references it is null. Values can be assigned during the declaration or within the constructor.
- Instance variables can be accessed directly by calling the variable name inside the class. However, within static methods (when instance variables are given accessibility), they should be called using the fully qualified name. *ObjectReference.VariableName*.

Class/static variables

- Class variables also known as static variables are declared with the static keyword in a class, but outside a method, constructor or a block.
- There would **only be one copy of each class variable per class**, regardless of how many objects are created from it.
- Static variables are rarely used other than being declared as constants. Constants are variables that are declared as public/private, final, and static. Constant variables never change from their initial value.

Class/static variables

- Visibility is similar to instance variables. However, most static variables are declared public since they must be available for users of the class.
- Default values are same as instance variables. For numbers, the default value is 0; for Booleans, it is false; and for object references, it is null. Values can be assigned during the declaration or within the constructor. Additionally, values can be assigned in special static initializer blocks.
- Static variables can be accessed by calling with the class name *ClassName.VariableName*.
- When declaring class variables as public static final, then variable names (constants) are all in upper case.

tutorialspoint  SIMPLY EASY LEARNING Javac 1.8.x

Execute | Main File

```
1  import java.io.*;
2
3  public class Employee{
4     // salary  variable is a private static variable
5     private static double salary;
6
7     // DEPARTMENT is a constant
8     public static final String DEPARTMENT = "Development ";
9
10    public static void main(String args[]){
11       salary = 1000;
12       System.out.println(DEPARTMENT + "average salary:" + salary);
13    }
14 }
```

Result

Compiling the source code....

$javac Employee.java 2>&1

Executing the program....

$java -Xmx128M -Xms16M Employee

```
Development average salary:1000.0
```

Java Modifiers

it is possible to modify classes, methods, etc., by using modifiers.

- **Access Modifiers: default, public , protected, private**.
- **Non-access Modifiers: static, final, abstract, synchronized, strictfp.**

Constant

- Được khai báo dùng từ khóa **final,** và thường dùng tiếp vĩ ngữ đối với các hằng số (l, L, d, D, f, F)
- Ví dụ:

  final int x = 10; // khai báo hằng số nguyên x = 10

  final long y = 20L; // khai báo hằng số long y = 20
- Hằng ký tự: đặt giữa cặp nháy đơn ‘’
- Hằng chuỗi: là một dãy ký tự đặt giữa cặp nháy đôi “”

Constant

Ký tự	Ý nghĩa
\b	Xóa lùi (BackSpace)
\t	Tab
\n	Xuống hàng
\r	Dấu enter
\”	Nháy kép
\’	Nháy đơn
\\	\
\f	Đẩy trang
\uxxxx	Ký tự unicode

Operators

- **Toán tử số học** (Arithmetic Operators)

Toán tử	Ý nghĩa
+	Cộng
-	Trừ
*	Nhân
/	Chia nguyên
%	Chia dư
++	Tăng 1
--	Giảm 1

Operators

- **Phép toán trên bit** (Bitwise Operators)

Toán tử	Ý nghĩa
&	AND
|	OR
^	XOR
<<	Dịch trái
>>	Dịch phải
~	Bù bit

Operators

- **Toán tử quan hệ & logic** (Relational Operators)

Toán tử	Ý nghĩa
= =	So sánh bằng
!=	So sánh khác
>	So sánh lớn hơn
<	So sánh nhỏ hơn
>=	So sánh lớn hơn hay bằng
<=	So sánh nhỏ hơn hay bằng
||	OR (biểu thức logic)
&&	AND (biểu thức logic)
!	NOT (biểu thức logic)

Operators

- **Toán tử gán (Assignment)**

Toán tử	Ví dụ	Ý nghĩa
=	a = b	gán a = b
+=	a += 5	a = a + 5
-=	b -= 10	b = b – 10
*=	c *= 3	c = c * 3
/=	d /= 2	d = d/2
%=	e %= 4	e = e % 4

Operators

- **Toán tử điều kiện (**Conditional Operator)

Cú pháp: *<điều kiện> ? <biểu thức 1> : < biểu thức 2>*

Ví dụ:

```
int x = 10;
int y = 20;
int Z = (x<y) ? 30 : 40;
// Kết quả z = 30 do biểu thức (x < y) là đúng.
```

Loop Control

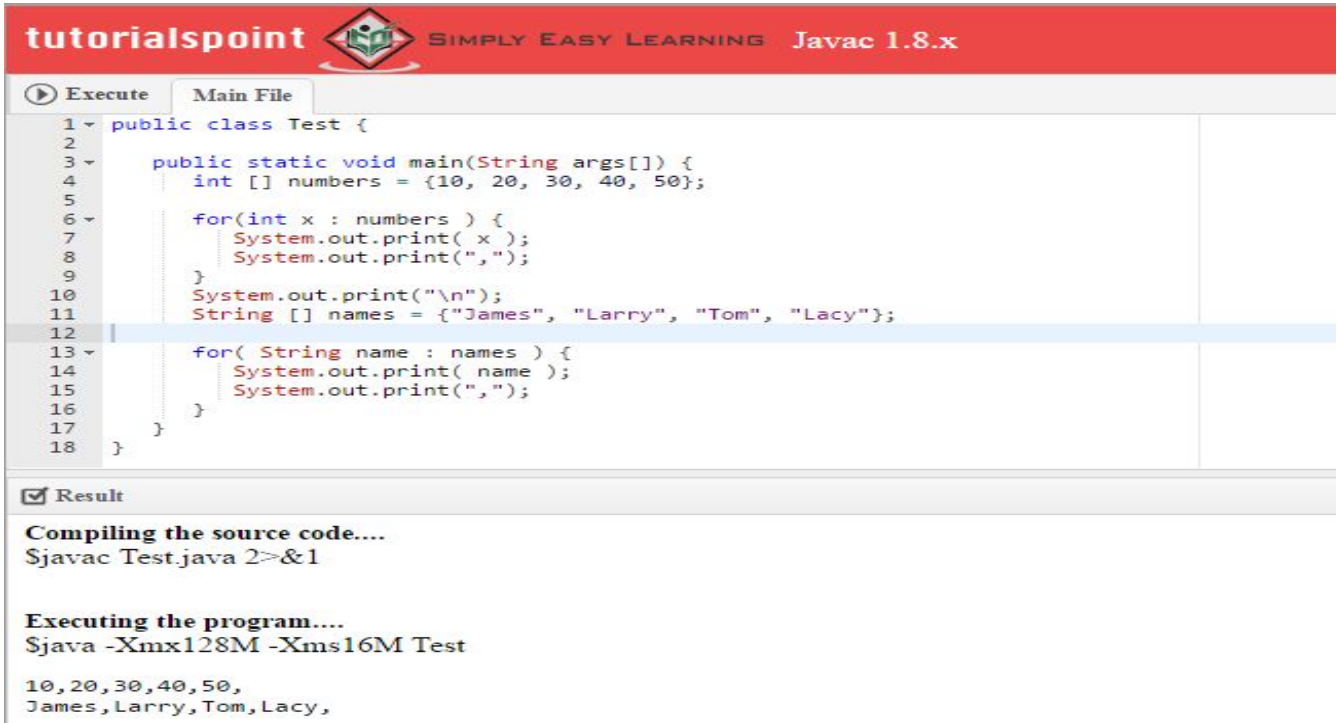
```
while( x < 20 ) {
   System.out.print("value of x : " + x );
   x++;
   System.out.print("\n");
}
```

```
do {
   System.out.print("value of x : " + x );
   x++;
   System.out.print("\n");
}while( x < 20 );
```

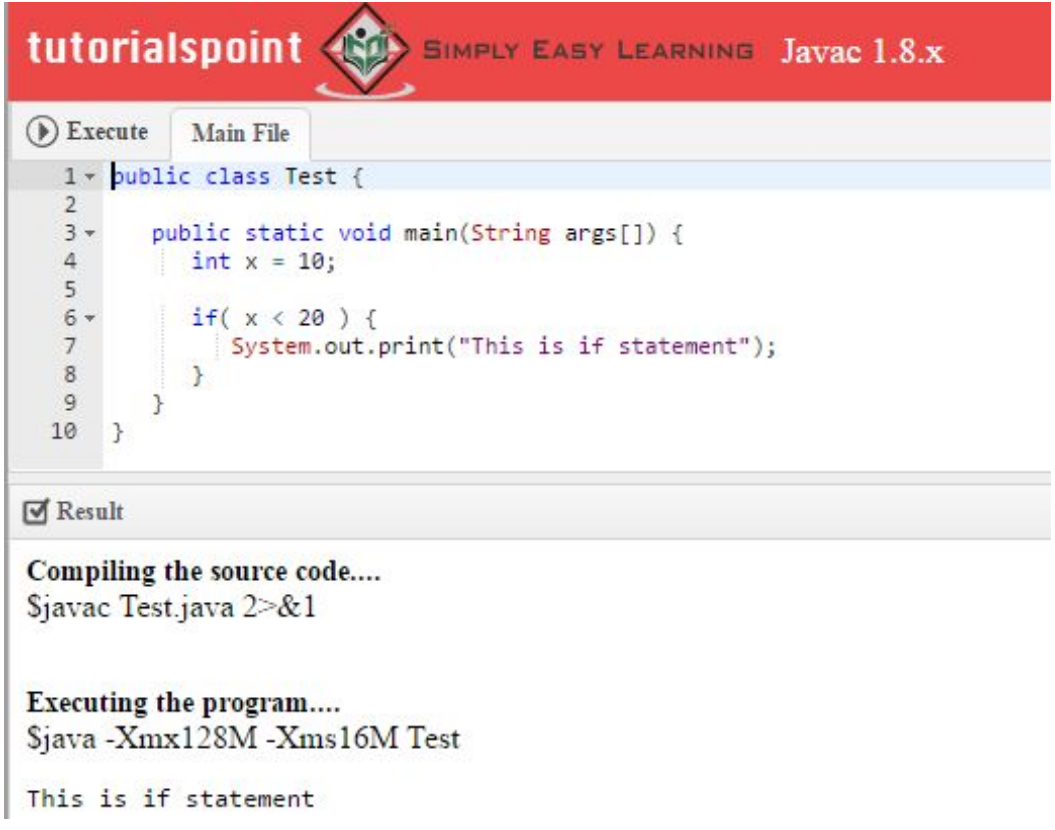
```
for(int x = 10; x < 20; x = x + 1) {
   System.out.print("value of x : " + x );
   System.out.print("\n");
}
```

Enhanced for loop in Java (Java 5)

Mainly used to traverse collection of elements including arrays.

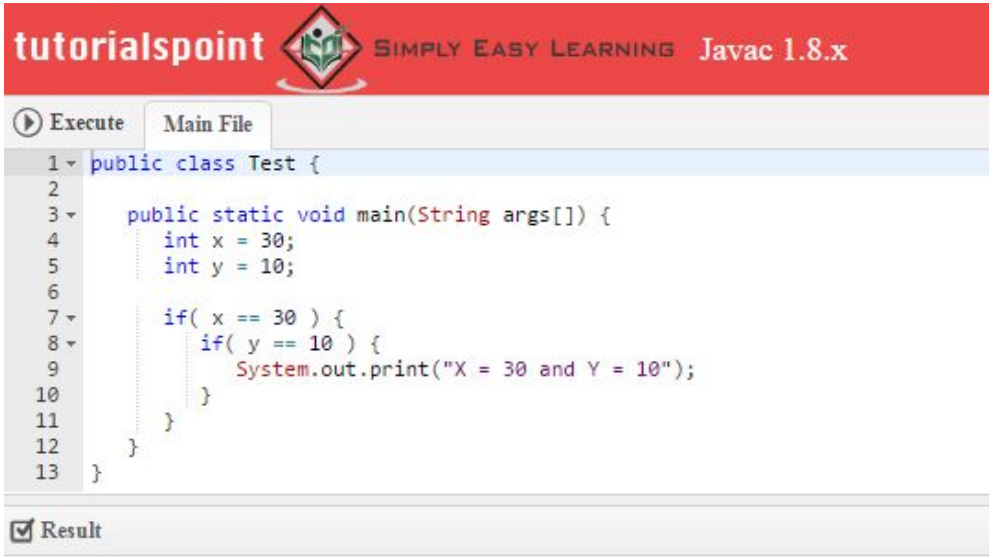


If statement



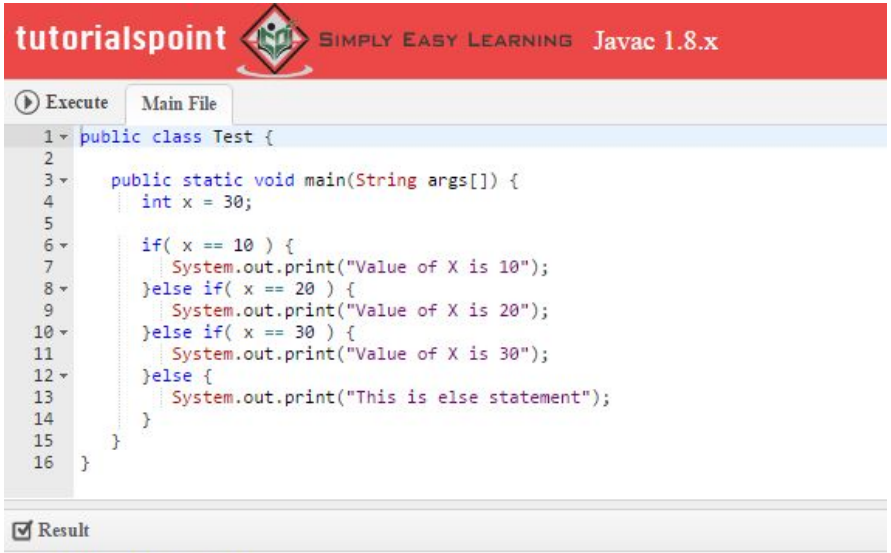
If statement

Nested if statement



If statement

The if...else if...else Statement



Switch statement

integers, byte, short, char, strings and enums.

Wrapper classes

in development, we come across situations where we need to use objects instead of primitive data types. In order to achieve this, Java provides**wrapper classes**.

Data Type	Wrapper classes (java.lang.)
byte	Byte
short	Short
int	Int
long	Long
float	Float
double	Double

Strings Class

- Strings, which are widely used in Java programming, are a sequence of characters.
- In Java, strings are treated as objects.

Creating Strings

- Whenever it encounters a string literal in your code

  String str = “Hello world”;

- Using the **new** keyword and a constructor.

Concatenating Strings

“Hello, ” + “world” + “!”;

“My name is ”.**concat**(“SEUIT”);

String Methods (JDK document)

https://docs.oracle.com/javase/7/docs/api/java/lang/String.html

Mathematical Functions

```
Math.sin(x);
Math.cos(Math.PI / 2);
Math.pow(2, 3);
Math.log(Math.log(x + y));
```

https://docs.oracle.com/javase/7/docs/api/java/lang/Math.html

Good Programming Style

- The goal of good style is to make your code more readable by you and by others.

Use good (meaningful) names

```
String a1;
int a2;
double b;                  // BAD!!

String firstName;          // GOOD
String lastName;           // GOOD
int temperature;           // GOOD
```

Naming

- **Case Sensitivity:** **Hello** and **hello** would have different meaning in Java.
- **Class Names:** the first letter should be in Upper Case. **E**ach inner word's first letter should be in Upper Case

  *class **M**y**F**irst**J**ava**C**lass*

Naming

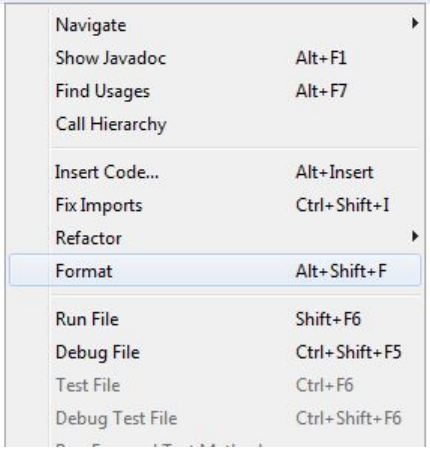
- **Method Names:** should start with a Lower Case letter.

  *public void **myMethodName()***

Use indentation

```
    public static void main(String[] arguments) {
        double x = 0;
        double y = 0;

        Math.sin(x);
        Math.cos(Math.PI / 2);
        Math.pow(2, 3);
        Math.log(Math.log(x + y));
    }
}
```



Use whitespaces

- Put whitespaces in complex expressions:

- Put blank lines to improve readability:

```
// BAD!!
double cel=fahr*42.0/(13.0-7.0);


// GOOD
double cel = fahr * 42.0 / (13.0 - 7.0);
```

Do not duplicate tests

```
if (basePay < 8.0) {
            ...
} else if (hours > 60) {
            ...
} else if (basePay >= 8.0 && hours <= 60){
            ...
}
```

BAD

Do not duplicate tests

```
if (basePay < 8.0) {
                ...
} else if (hours > 60) {
                ...
} else {
                ...
}
```

Good Programming Style

- Use good (meaningful) names
- Use indentation
- Use whitespaces
- Do not duplicate tests

Java Naming conventions

Name	Convention
class name	should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc.
interface name	should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc.
method name	should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc.
variable name	should start with lowercase letter e.g. firstName, orderNumber etc.
package name	should be in lowercase letter e.g. java, lang, sql, util etc.
constants name	should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc.

https://www.w3schools.com/java/exercise.asp?filename=exercise_syntax1