JAVA TUTORIAL

Instructor: DieuNT1

Table of contents

◊ **Introduction to Java**

◊ **First Java Program**

◊ **Basic Java Syntax**

◊ **Java Data Types**

◊ **Java Operators**

◊ **Variables and Constant**

Section 1

Introduction to Java

Introduction to Java

- **History:**
  - ✓ In 1991: OAK

  - ✓ A programming language that was introduced by Sun Microsystems in 1995, later acquired by **Oracle Corporation**.
    - Originally for intelligent consumer-electronic devices
    - Then used for creating Web pages with dynamic content

Introduction to Java (2)

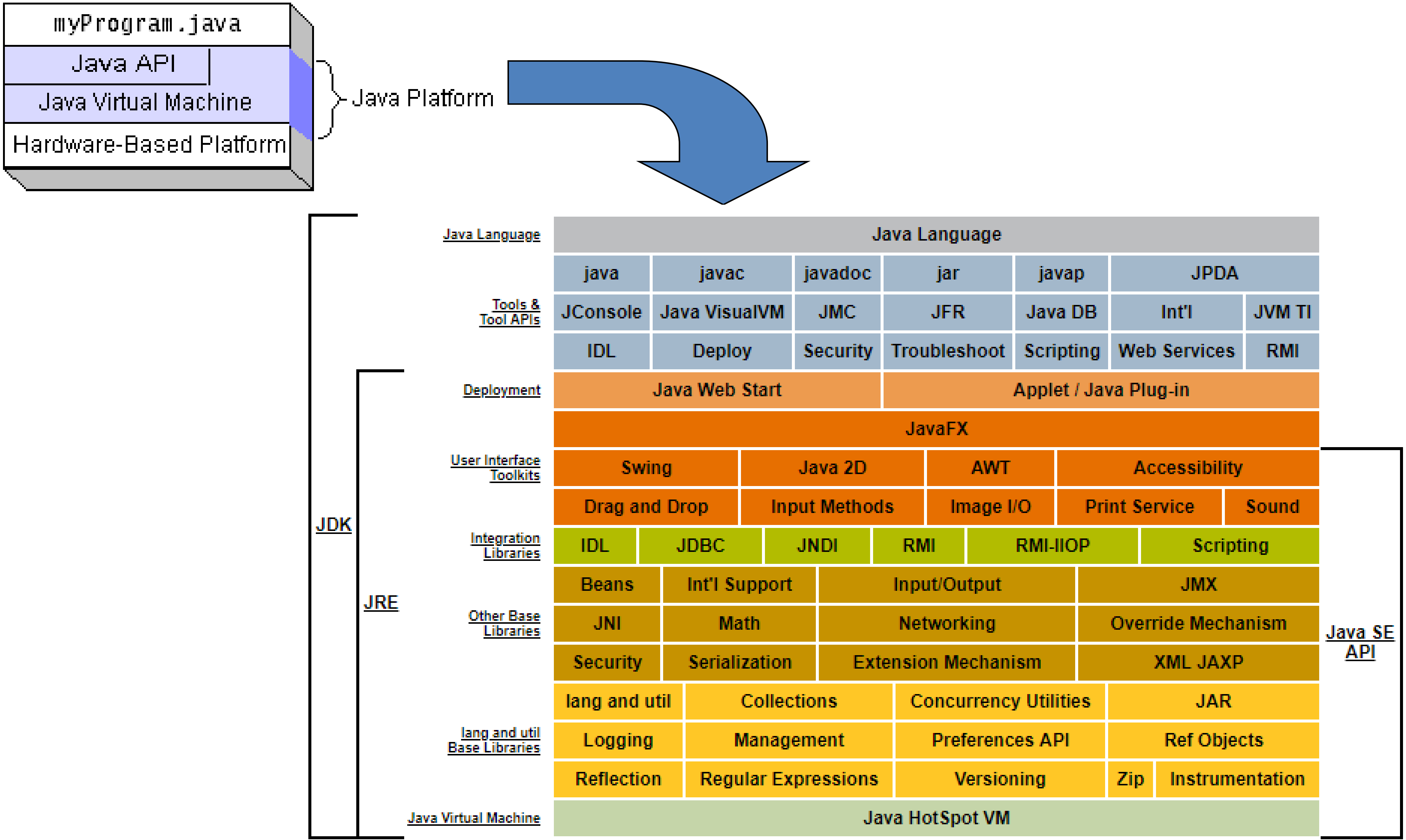
- **Now also used for:**
  - ✓ Develop large-scale enterprise applications
  - ✓ Enhance WWW server functionality
  - ✓ Provide applications for consumer[tiêu dùng] devices (cell phones, cloud, etc.)
- **Object-oriented programming**
- **Java Tutorial Online at**

  https://www.oracle.com/technetwork/java/javase/downloads/index.html

Main Features of JAVA

- The Java programming language is a high-level language that can be characterized by all of the following buzzwords:
  - ✓ **Simple**
  - ✓ **Object oriented**
  - ✓ **Distributed**
  - ✓ **Multithreaded**
  - ✓ **Dynamic**
  - ✓ **Architecture neutral**
  - ✓ **Portable**
  - ✓ **High performance**
  - ✓ **Robust**
  - ✓ **Secure**

Java Platform



myProgram.java
Java API
Java Virtual Machine
Hardware-Based Platform
Java Platform

JDK
JRE
Java SE API

Java Language: Java Language

Tools & Tool APIs:
java | javac | javadoc | jar | javap | JPDA
JConsole | Java VisualVM | JMC | JFR | Java DB | Int'l | JVM TI
IDL | Deploy | Security | Troubleshoot | Scripting | Web Services | RMI

Deployment: Java Web Start | Applet / Java Plug-in

User Interface Toolkits:
JavaFX
Swing | Java 2D | AWT | Accessibility
Drag and Drop | Input Methods | Image I/O | Print Service | Sound

Integration Libraries: IDL | JDBC | JNDI | RMI | RMI-IIOP | Scripting

Other Base Libraries:
Beans | Int'l Support | Input/Output | JMX
JNI | Math | Networking | Override Mechanism
Security | Serialization | Extension Mechanism | XML JAXP

lang and util Base Libraries:
lang and util | Collections | Concurrency Utilities | JAR
Logging | Management | Preferences API | Ref Objects
Reflection | Regular Expressions | Versioning | Zip | Instrumentation

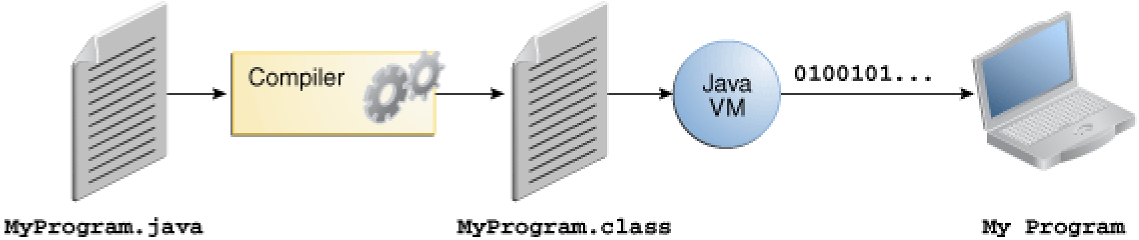
Java Virtual Machine: Java HotSpot VM

Java terminology

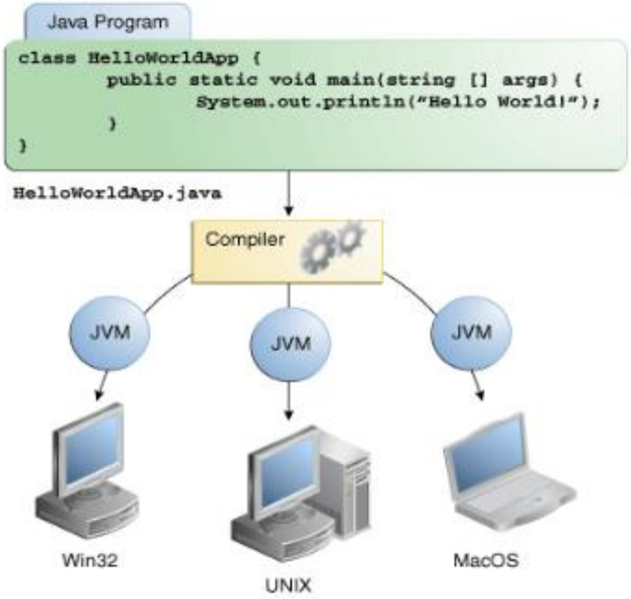
- **Java Development Kit(JDK)**
  - ✓ A complete java development kit that includes JRE (Java Runtime Environment), compilers and various tools like JavaDoc, Java debugger etc.
  - ✓ In order to create, compile and run Java program you would need JDK installed on your computer.
- **Java Runtime Environment(JRE)**
  - ✓ JRE is a part of JDK
  - ✓ When you have JRE installed on your system, you can run a java program however you won’t be able to compile it.
  - ✓ JRE includes JVM, browser plugins and applets support. When you only need to run a java program on your computer, you would only need JRE.

Java terminology

- **Java Virtual Machine (JVM)**



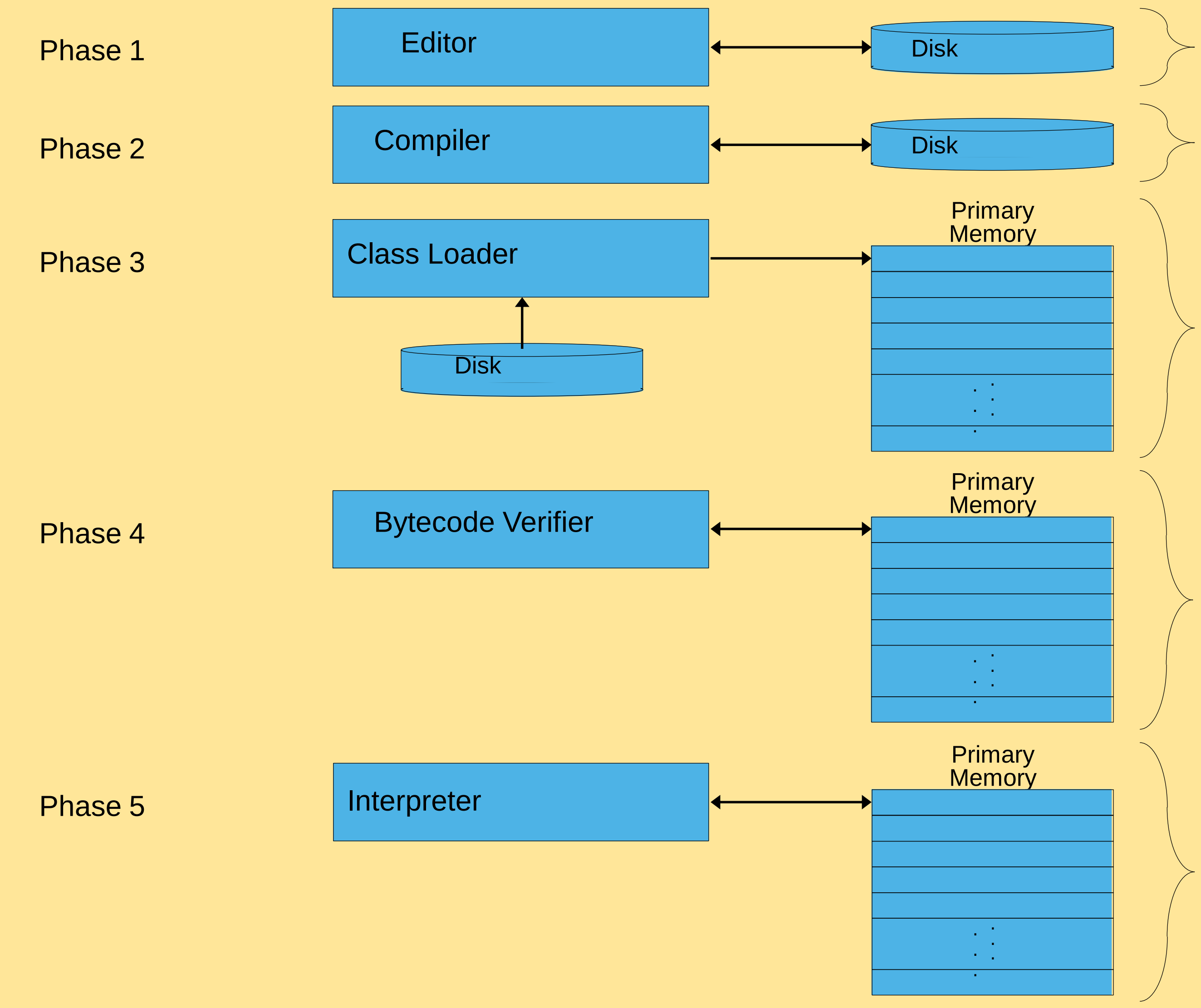
An overview of the software development process.



```
class HelloWorldApp {
        public static void main(string [] args) {
                System.out.println(“Hello World!”);
        }
}
```

Through the Java VM, the same application is capable of running on multiple platforms.

Basics of a Typical Java Environmen



Program is created in an editor and stored on disk in a file ending with `.java`.

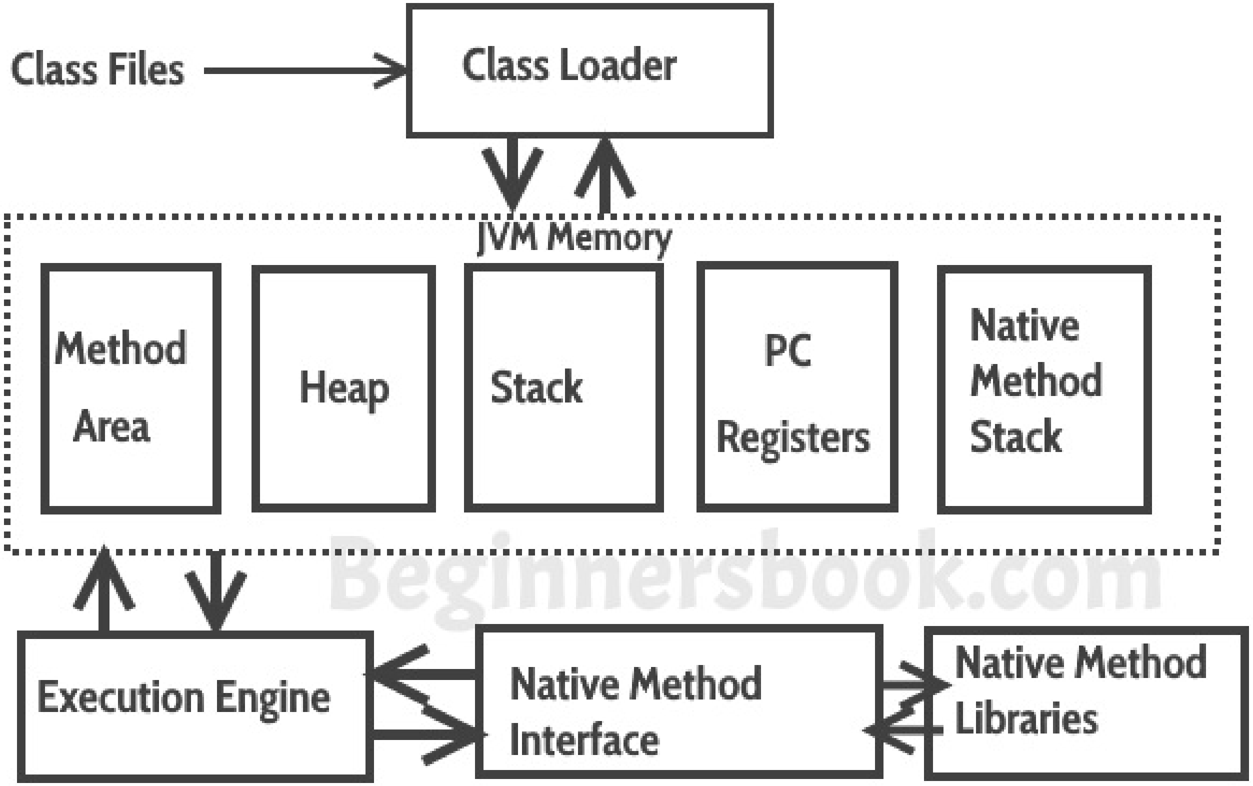
Compiler creates bytecodes and stores them on disk in a file ending with `.class`.

Class loader reads `.class` files containing bytecodes from disk and puts those bytecodes in memory.

Bytecode verifier confirms that all bytecodes are valid and do not violate[vi phạm] Java's security restrictions[giới hạn].

Interpreter reads bytecodes and translates them into a language that the computer can understand, possibly storing data values as the program executes.

JVM Architecture



JVM Architecture

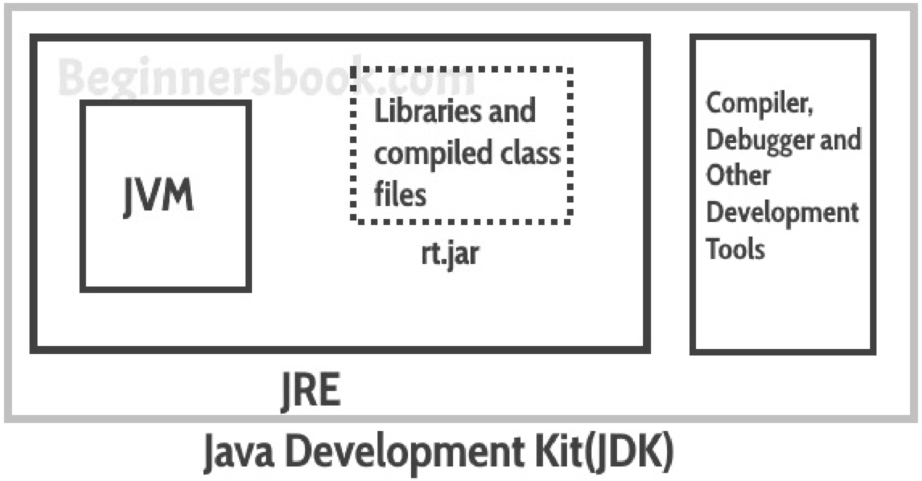
- **Class Loader:** The class loader reads the *.class* file and save the byte code in the **method area**.
- **Method Area**: There is only one method area in a JVM which is shared among all the classes. This holds the class level information of each *.class* file.
- **Heap**: Heap is a part of JVM memory where objects are allocated. JVM creates a Class object for each *.class* file.
- **Stack**: Stack is a also a part of JVM memory but unlike Heap, it is used for storing temporary variables.
- **PC Registers**: This keeps the track of which instruction$^{\text{[câu lệnh]}}$ has been executed and which one is going to be executed. Since instructions are executed by threads, each thread has a separate PC register.

JVM Architecture

- **Native Method stack:** A native method can access the runtime data areas of the virtual machine.
- **Native Method interface**: It enables java code to call or be called by native applications. Native applications are programs that are specific to the hardware and OS of a system.
- **Garbage collection**: A class instance is explicitly created by the java code and after use it is automatically destroyed by garbage collection for memory management.

JVM

- **Difference JDK, JRE & JVM?**
  - ✓ **JRE**: JRE is the environment within which the java virtual machine runs. JRE contains Java virtual Machine(JVM), class libraries, and other files excluding development tools such as compiler and debugger.
  - ✓ **JVM:** JVM runs the program by using class, libraries and files provided by JRE.
  - ✓ **JDK**: JDK is a superset of JRE, it contains everything that JRE has along with development tools such as compiler, debugger etc.


Beginnersbook.com
JVM
Libraries and compiled class files
rt.jar
Compiler, Debugger and Other Development Tools
JRE
Java Development Kit(JDK)

Section 2

First Java Program

First Sample: Printing a Line of Text

```java
//This is a simple program called First.java
public class First {
    public static void main(String[] args) {
    System.out.println("My first program in Java ");
    }
}
```

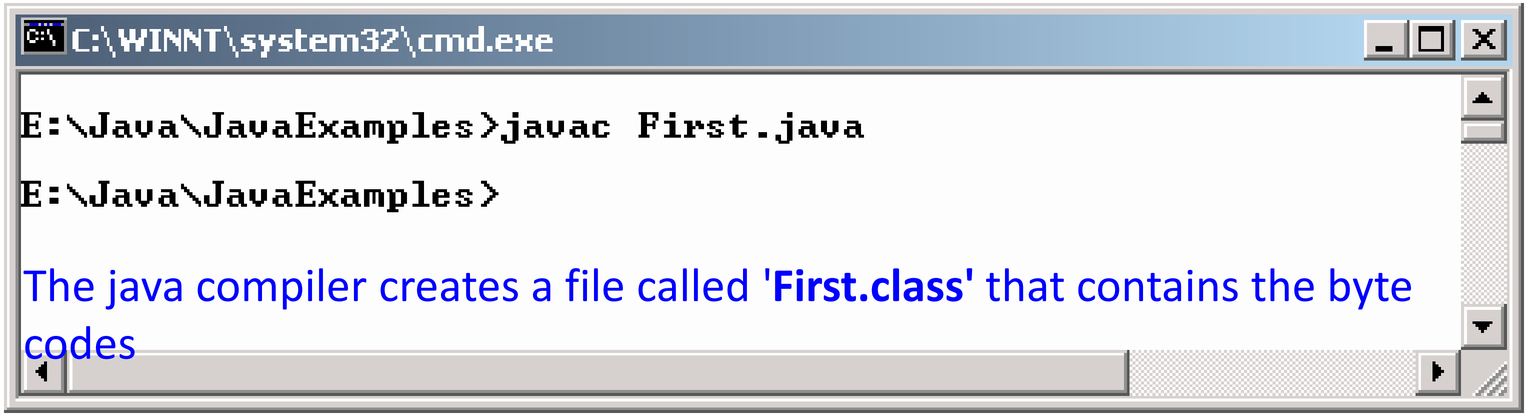
First Sample: Analyzing the Java Program

- **In which:**
  - ✓ The symbol `//` stands for commented line.
  - ✓ The line `class First` declares a new class called **`First`**.
  - ✓ `public static void main(String[] args)`

    This is the main method from where the program begins its execution.
  - ✓ `System.out.println("My first program in Java ");`

    This line displays the string **My first program in java** on the screen.

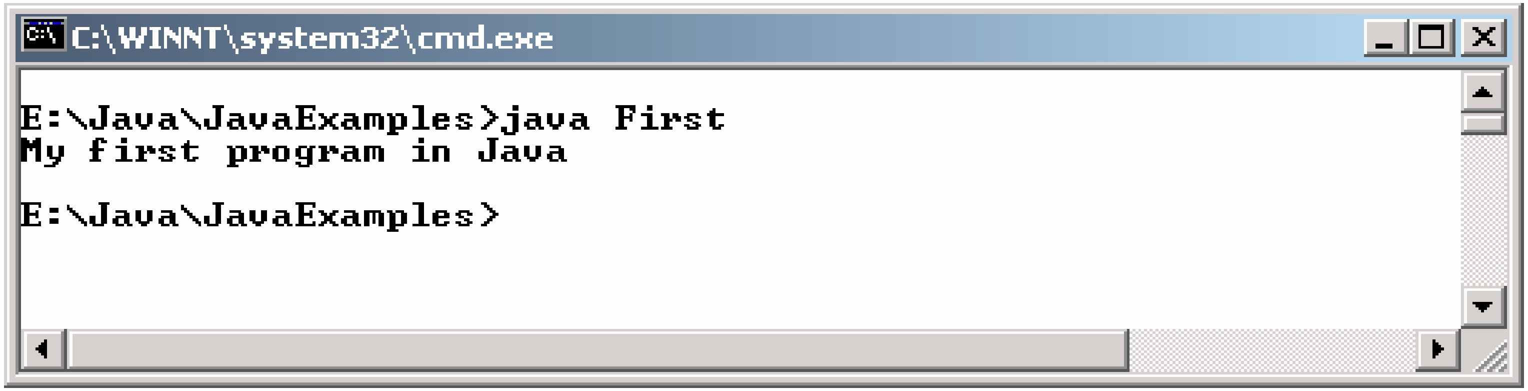
Compiling and executing



```
E:\Java\JavaExamples>javac First.java

E:\Java\JavaExamples>
```

The java compiler creates a file called '**First.class**' that contains the byte codes

To actually run the program, a java interpreter called `java` is required to execute the code.

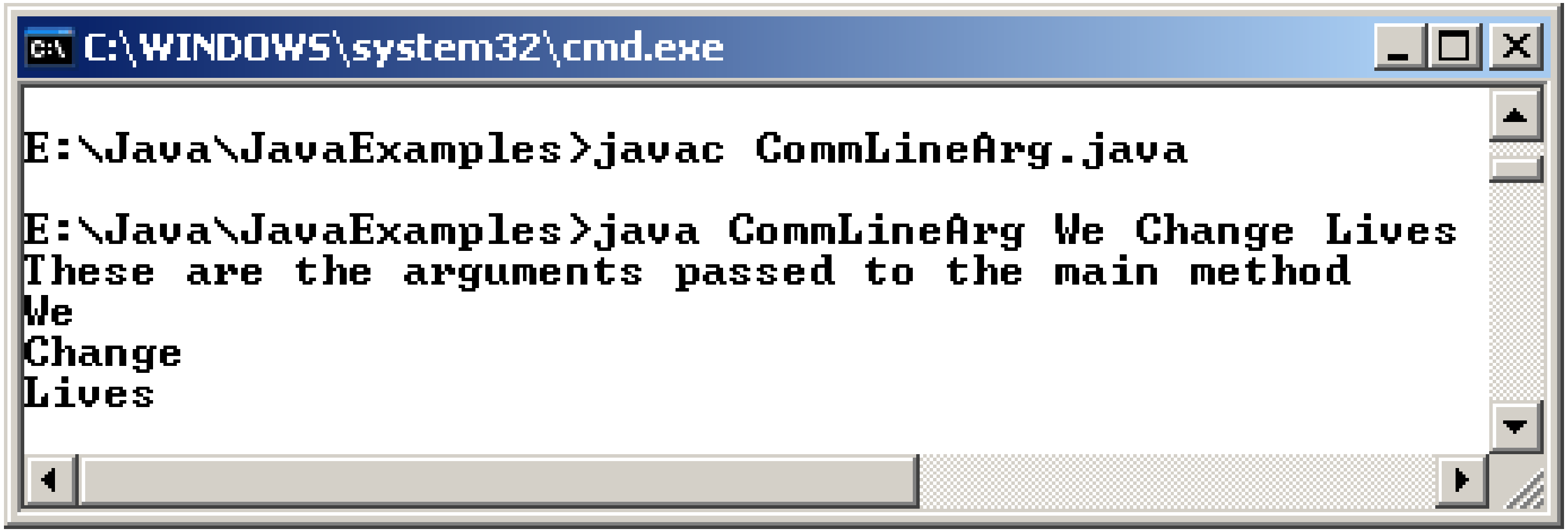


```
E:\Java\JavaExamples>java First
My first program in Java

E:\Java\JavaExamples>
```

Passing Command Line Arguments

```
public class CommLineArg {
    public static void main(String[] pargs) {
        System.out.
        println("These are the arguments passed to the
          main method.");
        System.out.println(pargs[0]);
        System.out.println(pargs[1]);
        System.out.println(pargs[2]);
    }
}
```

Passing Command Line Arguments



```
E:\Java\JavaExamples>javac CommLineArg.java

E:\Java\JavaExamples>java CommLineArg We Change Lives
These are the arguments passed to the main method
We
Change
Lives
```

Section 3

Basic Java Syntax

Code Comment

```
/*
 * Multi line
 */

// Single line

/**
 * Special comment for Javadocs
 */
```

Name Styles

- In Java, names are case-insensitive, may contains **letter**, **number**, the dollar sign "**$**", or the underscore character "_".
- Some convention name styles:
  - ✓ Class names: `CustomerInfo`
  - ✓ Variable, function names: `basicAnnualSalary`
  - ✓ Constants name: `MAXIMUM_NUM_OF_PARTICIPANTS`

Name Styles: Naming best practice

- Name should be **meaningful**
- Avoid very sort name, except for temporary "throwaway" variables: a, i, j
- Avoid confuse name: `TransferAction` class and `DoTransferAction` class, so which one will really performs the action?
- Class name should be a noun, use whole words, avoid acronyms and abbreviations: `Student`
- Variable name should begin with a noun: `numberOfFiles`
- Variable names should not start with underscore ('_') or dollar sign ('$') characters, even though both are allowed.
- Distinguish singular - plural: `Student - Students`
- Method name should begin with verb: `countNumberOfFiles()`
- As **clear** as possible: `annualSalary` instead of `salary`
- Avoid mixed-language, ex Vietnamese + English + Japanese.

Java Keywords

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

* not used

** added in 1.2

*** added in 1.4

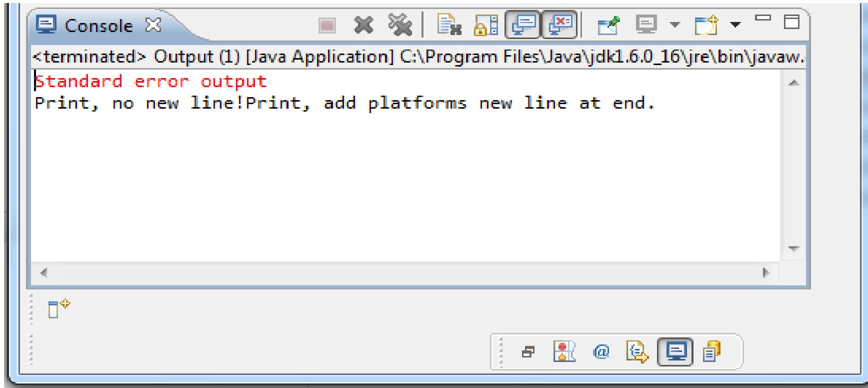
**** added in 5.0

true, false, and null might seem like keywords, but they are actually literals; you cannot use them as identifiers in your programs.

Standard Java Output

- `System.out` is standard out in Java
- `System.err` is error out in Java
- **Ex:**

```
public class Output {
    public static void main(String[] args) {
    System.out.print("Print, no new line!");
    System.out.println("Print, add platforms new line at
      end.");
    System.out.flush();
    System.err.println("Standard error output");
    }
}
```



```
<terminated> Output (1) [Java Application] C:\Program Files\Java\jdk1.6.0_16\jre\bin\javaw.
Standard error output
Print, no new line!Print, add platforms new line at end.
```

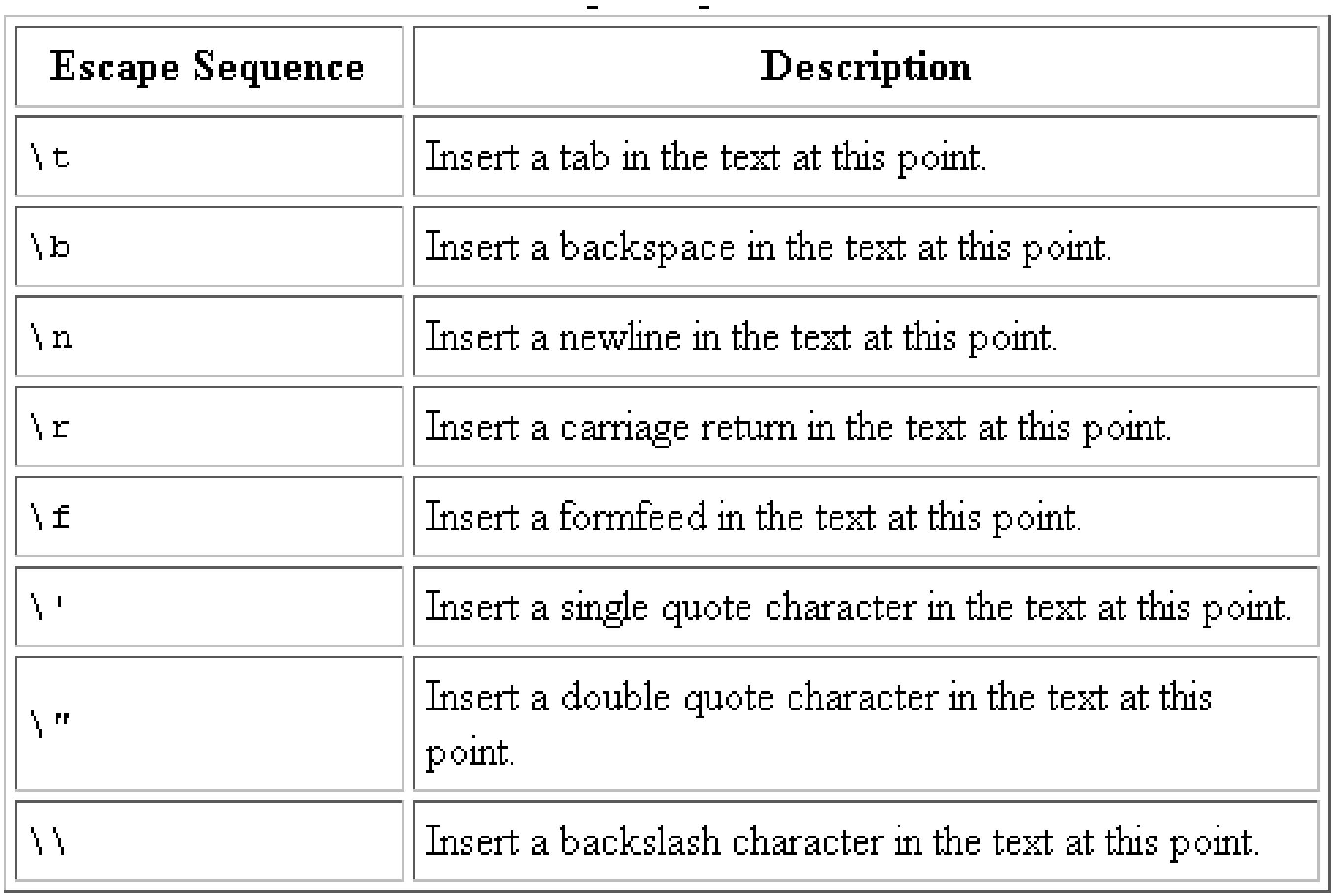
Standard Java Input

- `System.in` is standard input in Java
- The following program reads characters from the keyboard then print out to the screen.

```java
public class Echo {
    public static void main(String[] args) throws
      IOException{
        int ch;
        System.out.println("Enter some text: ");
        while ((ch = System.in.read()) != '\n') {
          System.out.print((char) ch);
        }
    }
}
```

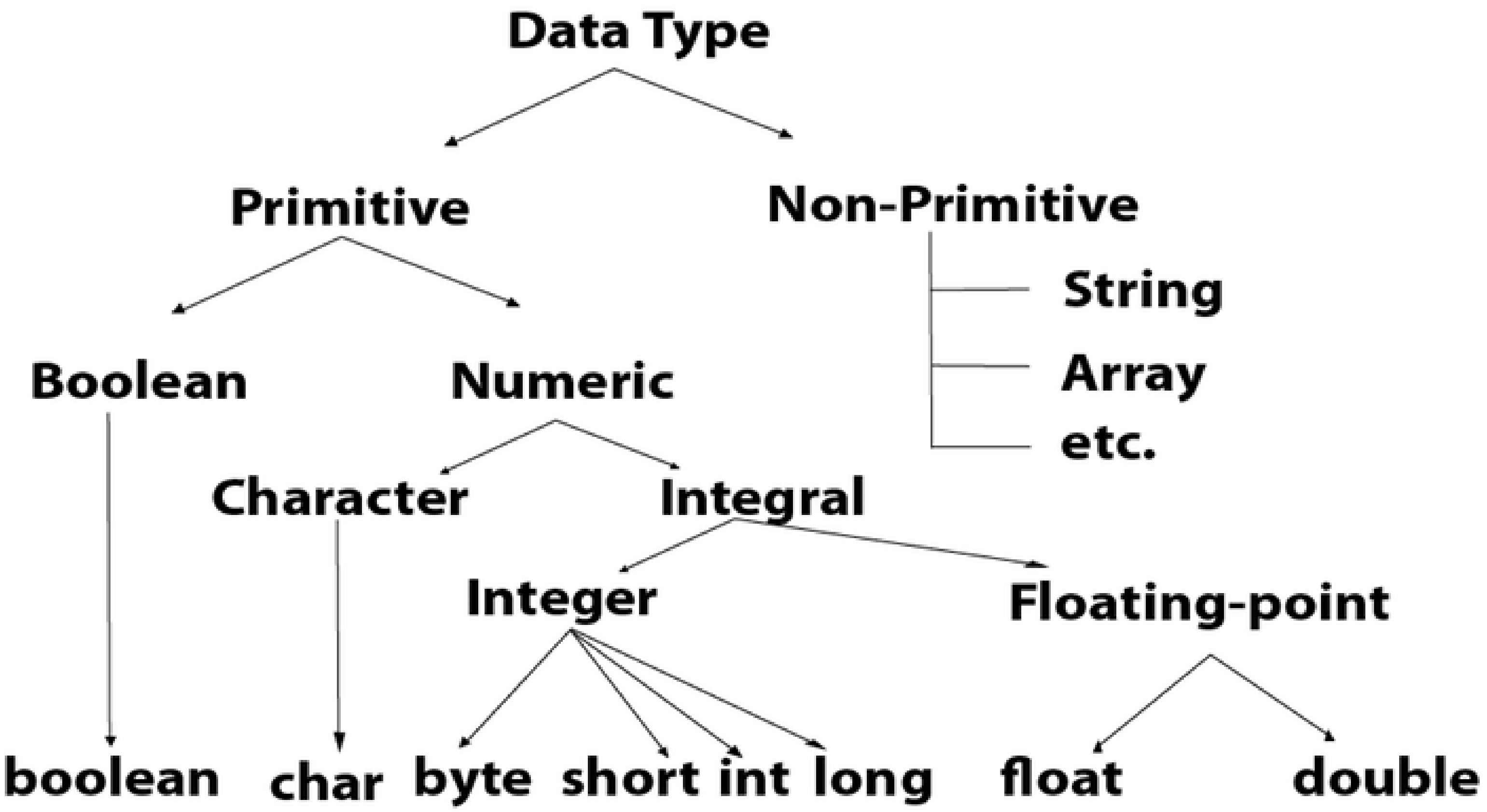
Escape characters

- Escape characters is backslash (\ )



Escape Sequence	Description
\t	Insert a tab in the text at this point.
\b	Insert a backspace in the text at this point.
\n	Insert a newline in the text at this point.
\r	Insert a carriage return in the text at this point.
\f	Insert a formfeed in the text at this point.
\'	Insert a single quote character in the text at this point.
\"	Insert a double quote character in the text at this point.
\\	Insert a backslash character in the text at this point.

Basic Data Types



Basic Data Types

- **byte**: The byte data type is an **8-bit** signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive).
- **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive)
- **int**: The int data type is a 32-bit signed two's complement integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive).
- **long**: The long data type is a 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807 (inclusive)

Basic Data Types (2)

- **float**: The float data type is a single-precision 32-bit IEEE 754 floating point. Its range of values is from 3.4E^{-45} to 3.4E^{38}
- **double**: The double data type is a double-precision 64-bit IEEE 754 floating point. Its range of values is from 1.7E^{-324} to 1.7976931348623157E^{308}
- **boolean**: The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.
- **char**: The char data type is a single 16-bit Unicode character. It has a minimum value of '¥u0000' (or 0) and a maximum value of '¥uffff' (or 65,535 inclusive).

Basic Data Types (3)

- **Default Values**
  - ✓ It's not always necessary to assign a value when a field is declared
  - ✓ Fields that are declared but not initialized will be set to a reasonable default by the compiler
  - ✓ Generally speaking, this default will be **zero** or **null**, depending on the data type. However, **is generally considered bad programming style**.

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Section 4

Operators

Operators

- **Simple Assignment Operator**

=	Simple assignment operator

- **Arithmetic Operators**

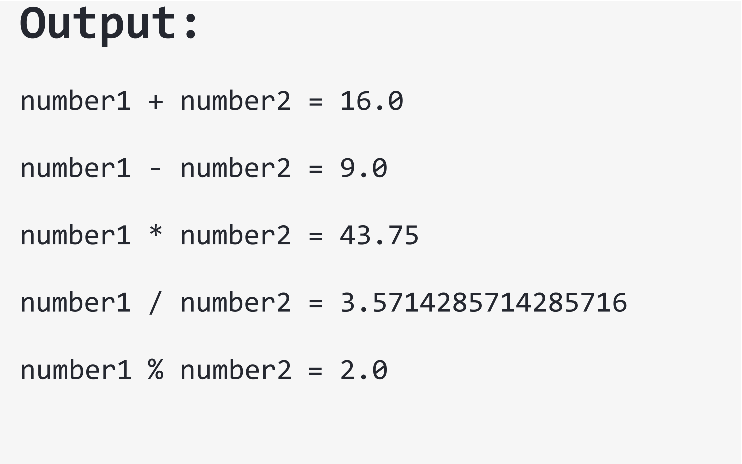
+	Additive operator
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Remainder operator

- **Unary Operators**

+	Unary plus operator; indicates positive value
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical compliment operator; inverts the value of a boolean

Operators

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

    double number1 = 12.5, number2 = 3.5, result;

    // Using addition operator
    result = number1 + number2;
    System.out.println("number1 + number2 = " + result);

    // Using subtraction operator
    result = number1 - number2;
    System.out.println("number1 - number2 = " + result);

    // Using multiplication operator
    result = number1 * number2;
    System.out.println("number1 * number2 = " + result);

    // Using division operator
    result = number1 / number2;
    System.out.println("number1 / number2 = " + result);

    // Using remainder operator
    result = number1 % number2;
    System.out.println("number1 % number2 = " + result);
  }
}
```



Output:

```
number1 + number2 = 16.0

number1 - number2 = 9.0

number1 * number2 = 43.75

number1 / number2 = 3.5714285714285716

number1 % number2 = 2.0
```

Operators

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

    double number = 5.2;
    boolean flag = false;

    System.out.println("+number = " + +number);
    // number is equal to 5.2 here.

    System.out.println("-number = " + -number);
    // number is equal to 5.2 here.

    // ++number is equivalent to number = number + 1
    System.out.println("number = " + ++number);
    // number is equal to 6.2 here.

    // -- number is equivalent to number = number - 1
    System.out.println("number = " + --number);
    // number is equal to 5.2 here.

    System.out.println("!flag = " + !flag);
    // flag is still false.
  }
}
```

Output:

```
+number = 5.2
-number = -5.2
number = 6.2
number = 5.2
!flag = true
```

Operators

- **Equality and Relational Operators**

==	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

- **Conditional Operators**

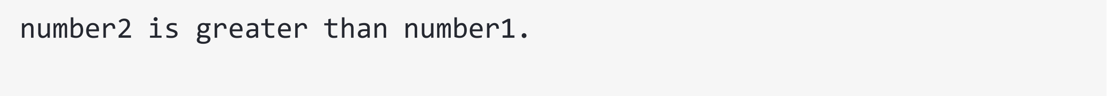
<table>
<tr><td>&&</td><td>Conditional-AND</td></tr>
<tr><td>||</td><td>Conditional-OR</td></tr>
<tr><td>?:</td><td>Ternary (shorthand for if-then-else statement)</td></tr>
</table>

- **Type Comparison Operator**

instanceof	Compares an object to a specified type

Operators

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

    int number1 = 5, number2 = 6;

    if (number1 > number2) {
      System.out.println("number1 is greater than number2.");
    } else {
      System.out.println("number2 is greater than number1.");
    }
  }
}
```



```
number2 is greater than number1.
```

```java
public class InstanceofOperator {
  public static void main(String[] args) {
    String test = "FPT";
    boolean result;

    result = test instanceof String;
    System.out.println(result);
  }
}
```

Operators

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

    int februaryDays = 29;
    String result;

    result = (februaryDays == 28) ? "Not a leap year" :
                                                   "Leap year";

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

```
Leap year
```

Operators

- **Bitwise and Bit Shift Operators**

~	Unary bitwise complement (đảo bít)
<<	Signed left shift
>>	Signed right shift
>>>	Unsigned right shift
&	Bitwise AND
^	Bitwise exclusive OR (triệt tiêu = XOR)
|	Bitwise inclusive OR

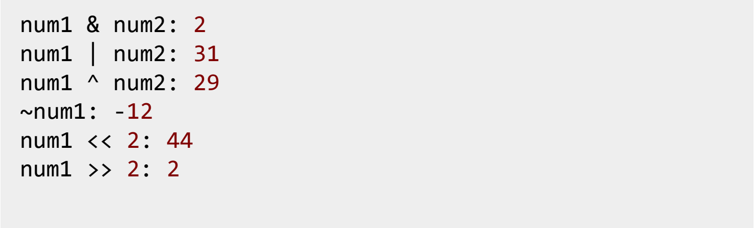
Operators

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

    int number1 = 1, number2 = 2, number3 = 9;
    boolean result;

    // At least one expression needs to be true for result to be true
    result = (number1 > number2) || (number3 > number1);
    // result will be true because (number1 > number2) is true
    System.out.println(result);

    // All expression must be true from result to be true
    result = (number1 > number2) && (number3 > number1);
    // result will be false because (number3 > number1) is false
    System.out.println(result);
  }
}
```

```
true false
```

Operators

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

    int num1 = 11; /* 11 = 00001011 */
    int num2 = 22; /* 22 = 00010110 */
    int result = 0;

    result = num1 & num2;
    System.out.println("num1 & num2: " + result);

    result = num1 | num2;
    System.out.println("num1 | num2: " + result);

    result = num1 ^ num2; //  generates 1 if they are not equal, else it returns 0.
    System.out.println("num1 ^ num2: " + result);

    result = ~num1;// changes the bit from 0 to 1 and 1 to 0.
    System.out.println("~num1: " + result);

    result = num1 << 2;
    System.out.println("num1 << 2: " + result);
    result = num1 >> 2;
    System.out.println("num1 >> 2: " + result);
  }
}
```



```
num1 & num2: 2
num1 | num2: 31
num1 ^ num2: 29
~num1: -12
num1 << 2: 44
num1 >> 2: 2
```

Operator Precedence

Operators	**Precedence**
postfix	*expr*++ *expr*--
unary	++*expr* --*expr* +*expr* -*expr* ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	|
logical AND	&&
logical OR	||
ternary	? :
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Type Casting

- In type casting, a data type is converted into another data type.
- **Automatic Type Promotion in Expressions**
- **Example:**

```
public class AutomaticTypePromotion {
    public static void main(String[] argv) {
        byte a = 40;
        byte b = 50;
        byte c = 100;
        int d = a * b / c;
        b = b * 2; // Error! Cannot assign an int to a byte!
        System.out.println("Value d: " + d);
    }
}
```

Type Casting

- **Type casting in Expressions**

  Casting is used for explicit type conversion. It loses information above the magnitude of the value being converted

- **Example:**

```
float f = 34.89675f;
d = (int) (f + 10);
```

Type Casting

- **Widening**[an toàn/mở rộng]**conversions:**

  char->int

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

- **Here are the Type Promotion Rules**
  - ✓ All `byte` and `short` values are promoted to `int` type.
  - ✓ If one operand is `long`, the whole expression is promoted to `long`.
  - ✓ If one operand is `float` then the whole expression is promoted to `float`.
  - ✓ If one operand is `double` then the whole expression is promoted to `double`.

Section 5

Variable and Constant

Variables and constants

- **Variable:**
- Three components of a variable declaration are:
  - ✓ Data type
  - ✓ Name
  - ✓ Initial value to be assigned (optional)
- **Syntax**

```
datatype identifier [=value][, identifier[=value]...];
```

- **Example:**

```
int foo = 42;
double d1 = 3.14, d2 = 2 * 3.14;
boolean isFun = true;
```

Variables and constants

- **Constants:**
  - ✓ It makes code more readable
  - ✓ It saves work when you make a change
  - ✓ You avoid risky[rủi ro] errors
  - ✓ In the case of string text
- **Syntax**

```
static final datatype CONSTNAME = value;
```

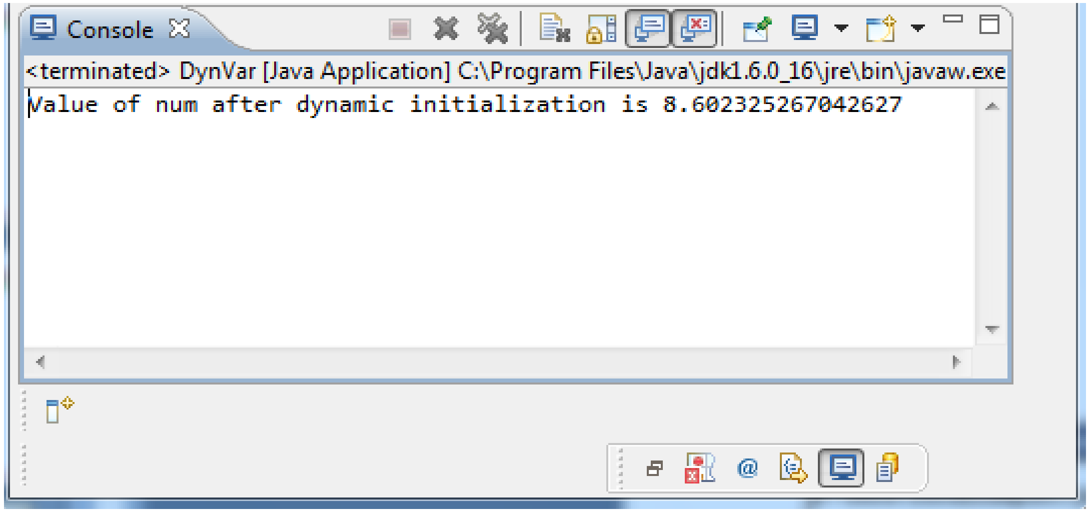
- **Example:**

```
static final int MAX_SECONDS = 25;
static final float PI = 3.14f;
```

Variables and constants

- **Example:**

```
public class DynVar {
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        double len = 5.0, wide = 7.0;
        double num = Math.sqrt(len * len + wide * wide);
        System.out.println("Value of num after dynamic
            initialization is " + num);
    }
}
```



```
<terminated> DynVar [Java Application] C:\Program Files\Java\jdk1.6.0_16\jre\bin\javaw.exe
Value of num after dynamic initialization is 8.602325267042627
```

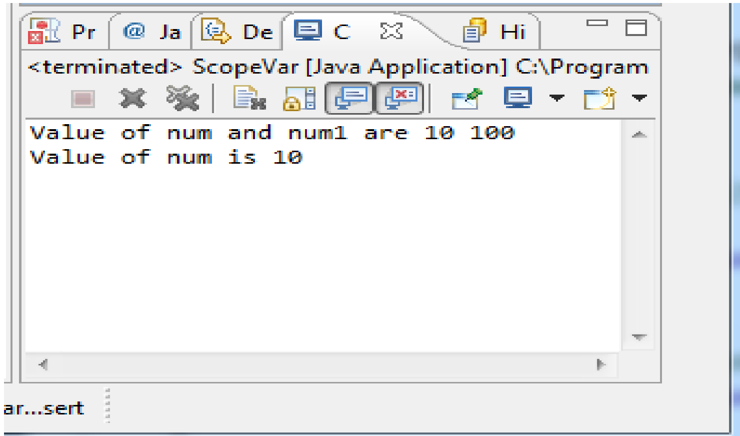
Scope and Lifetime of Variables

- Variables can be declared inside a block.
  - ✓ The block begins with an opening curly brace and ends with a closing curly brace.
  - ✓ A block defines a scope.
  - ✓ A new scope is created every time a new block is created.
- Scope specifies what objects are **visible** to other parts of the program.
- It also determines the life of an object.

Scope and Lifetime of Variables

- **Example:**

```
public class ScopeVar {
 public static void main(String[] args) {
     // TODO Auto-generated method stub
     int num = 10;
     if (num == 10) {
     // num is available in inner scope
     int num1 = num * num;
     System.out.println("Value of num and num1 are " + num + " "
       + num1);
     }
     // num1 = 10; ERROR ! num1 is not known
     System.out.println("Value of num is " + num);
 }
}
```



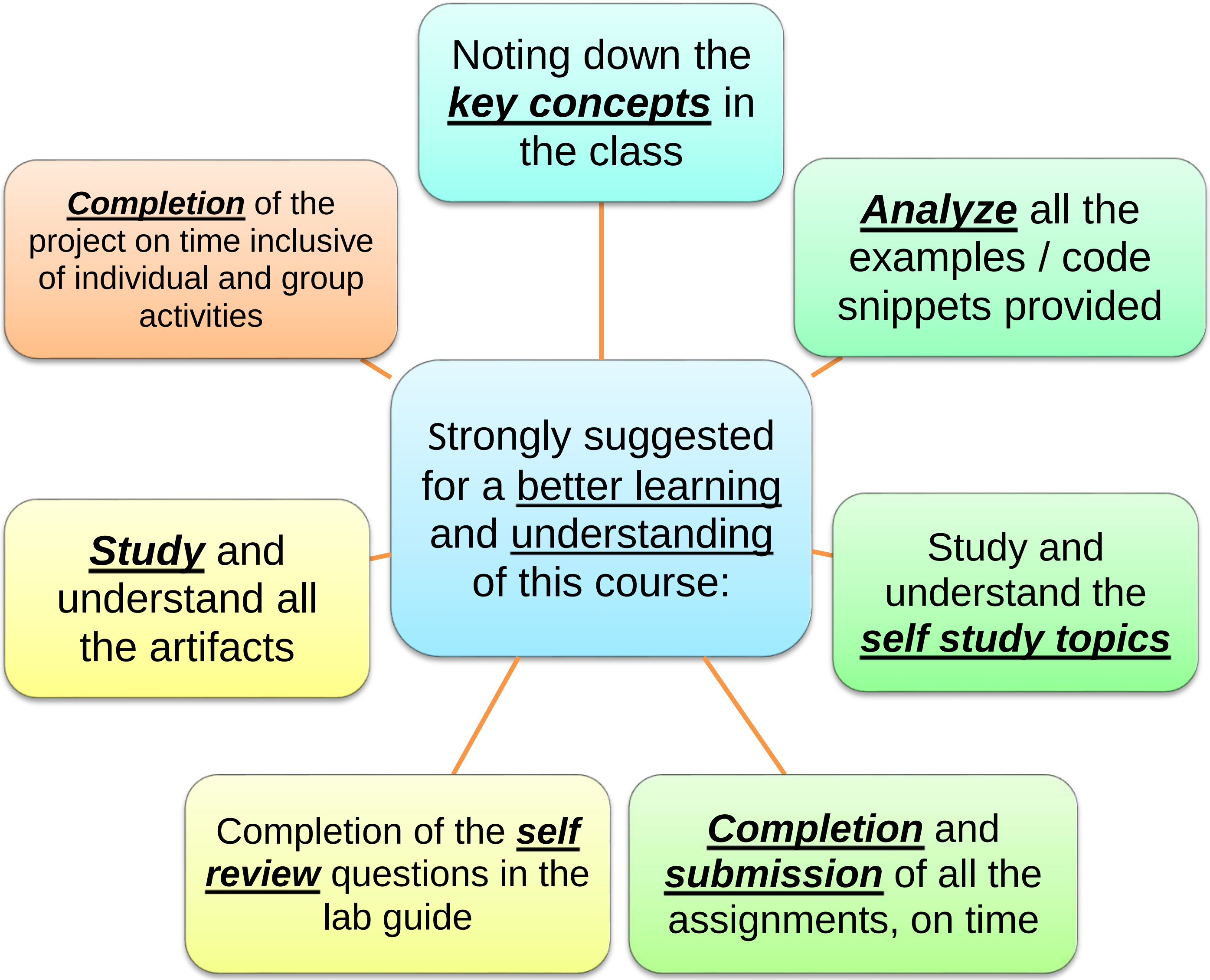
<terminated> ScopeVar [Java Application] C:\Program

```
Value of num and num1 are 10 100
Value of num is 10
```

SUMMARY

- ◊ Introduction to Java
- ◊ First Java Program
- ◊ Basic Java Syntax
- ◊ Java Data Types
- ◊ Java Operators
- ◊ Variables and Constant

Learning Approach



Strongly suggested for a better learning and understanding of this course:

Noting down the ***key concepts*** in the class

Analyze all the examples / code snippets provided

Study and understand the ***self study topics***

Completion and ***submission*** of all the assignments, on time

Completion of the ***self review*** questions in the lab guide

Study and understand all the artifacts

Completion of the project on time inclusive of individual and group activities

Thank you