

JAVA INTRODUCTION

Instructor: DieuNT1



Agenda

01. Introduction to Java

02. First Java Program

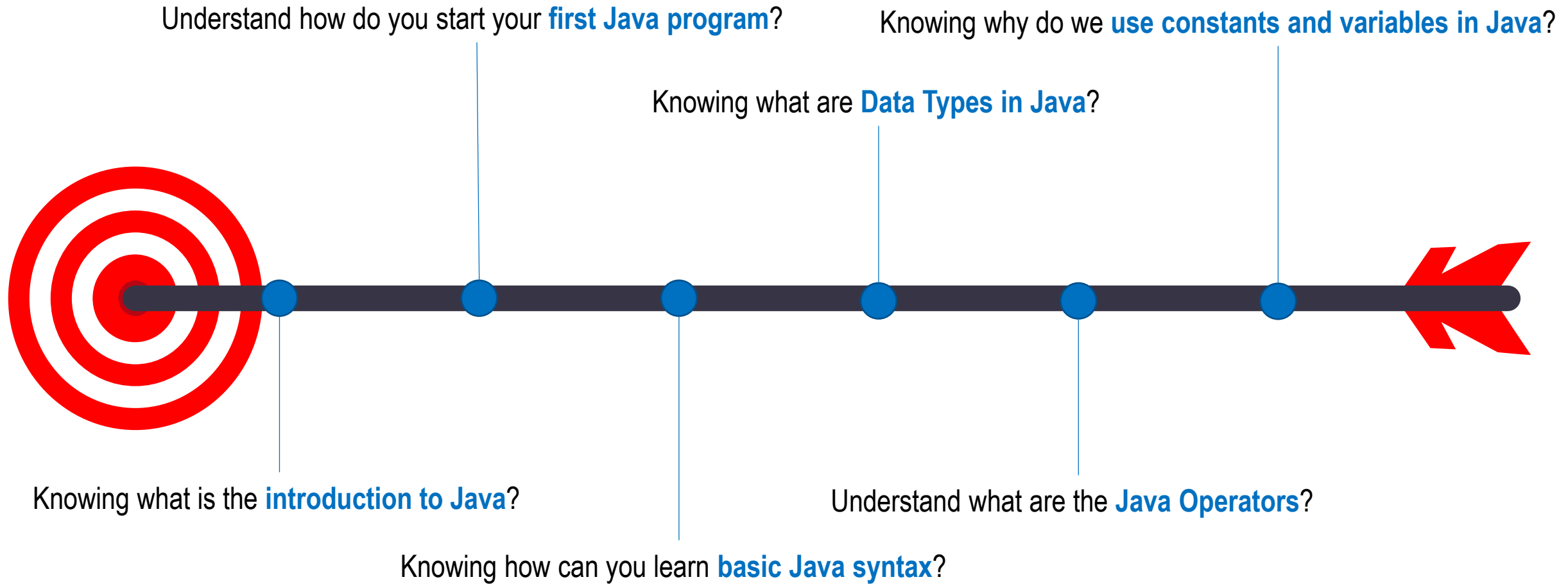
03. Basic Java Syntax

04. Java Data Types

05. Java Operators

06. Variables and Constant

Lesson Objectives





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**

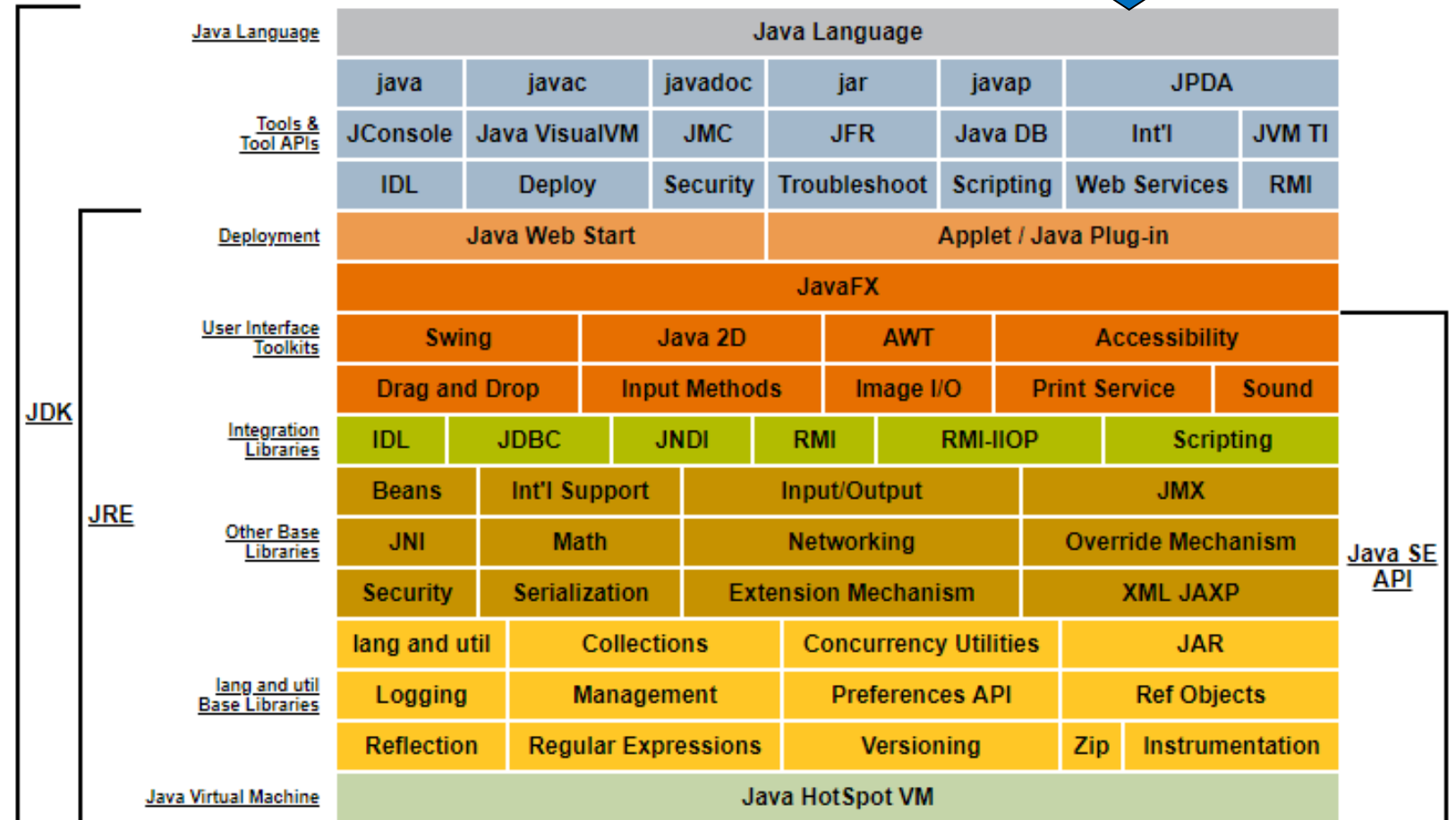
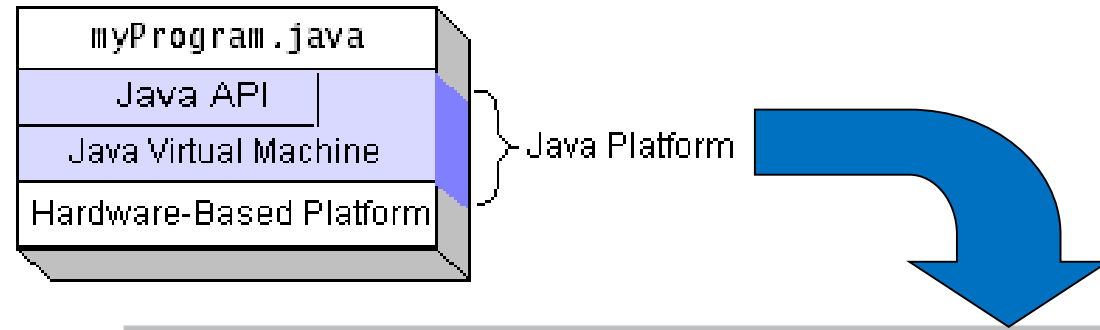
- **Java Tutorial Online at**

<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



Java terminology

Java Development Kit(JDK)

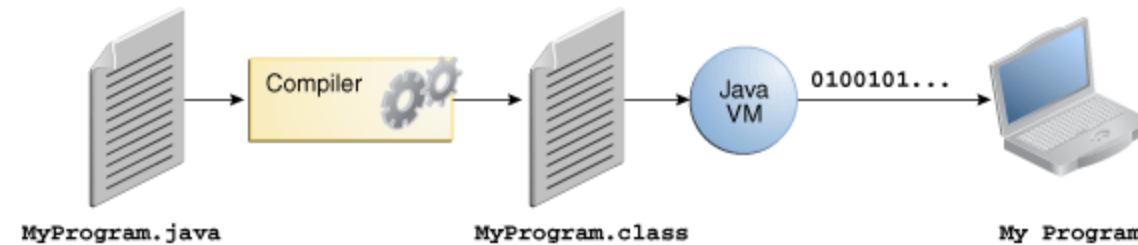
- A complete java development kit that includes JRE (Java Runtime Environment), compilers and various tools like JavaDoc, Java debugger etc.
- In order to create, compile and run Java program you would need JDK installed on your computer.

Java Runtime Environment(JRE)

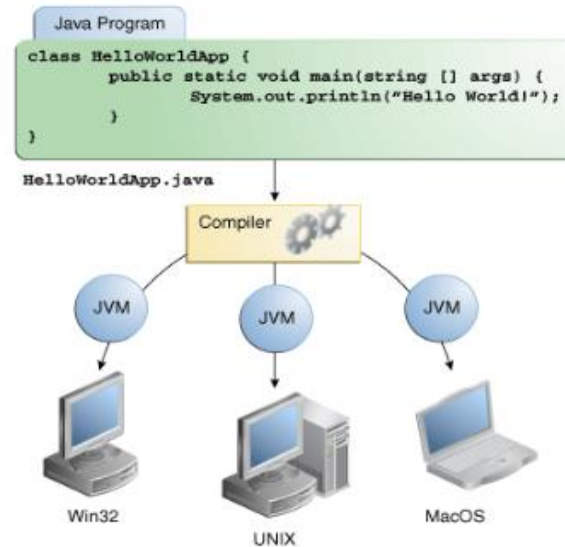
- JRE is a part of JDK
- When you have JRE installed on your system, you can run a java program however you won't be able to compile it.
- JRE includes JVM, browser plugins and applets support. When you only need to run a java program on your computer, you would only need JRE.

Java terminology

▪ Java Virtual Machine (JVM)

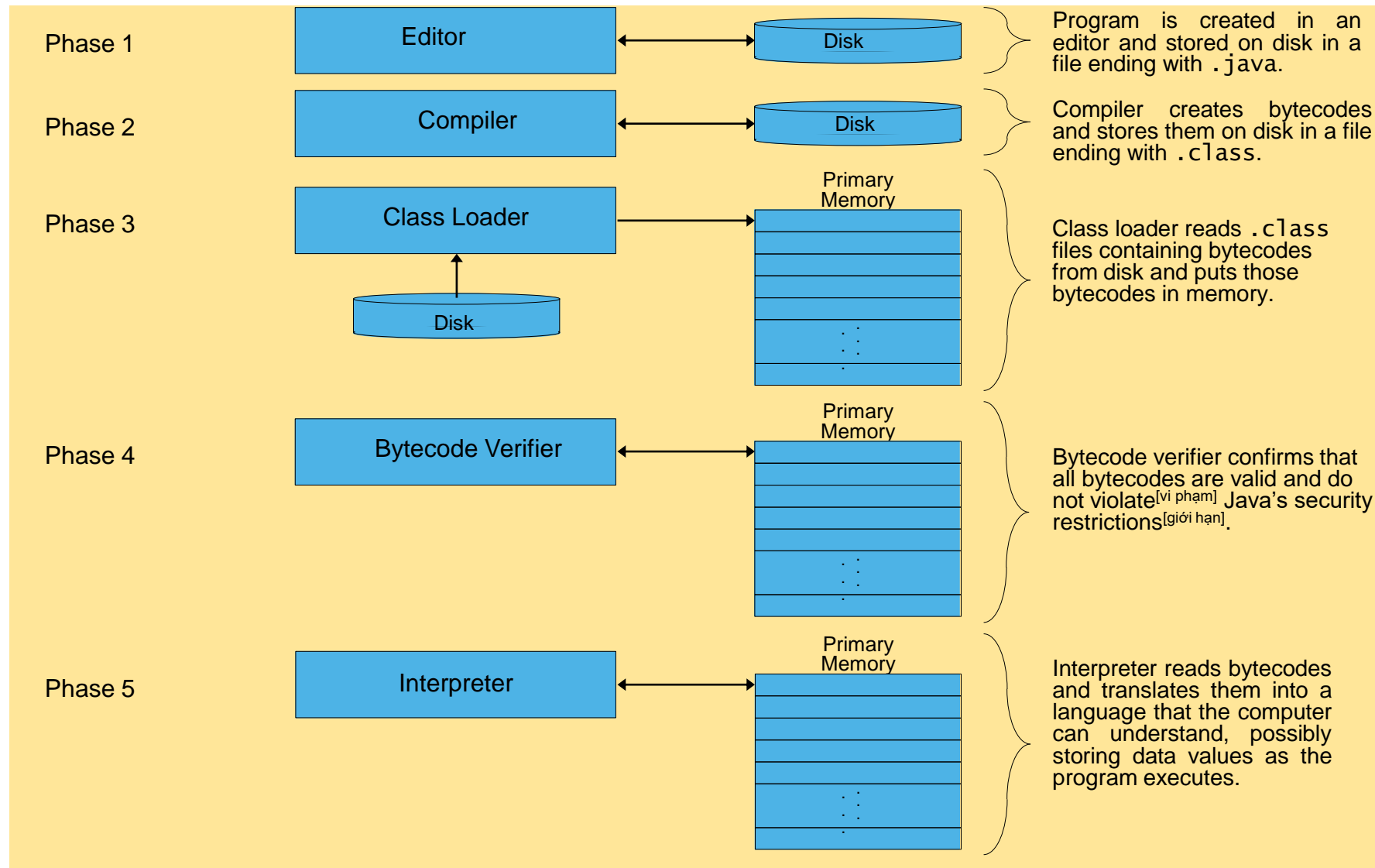


An overview of the software development process.

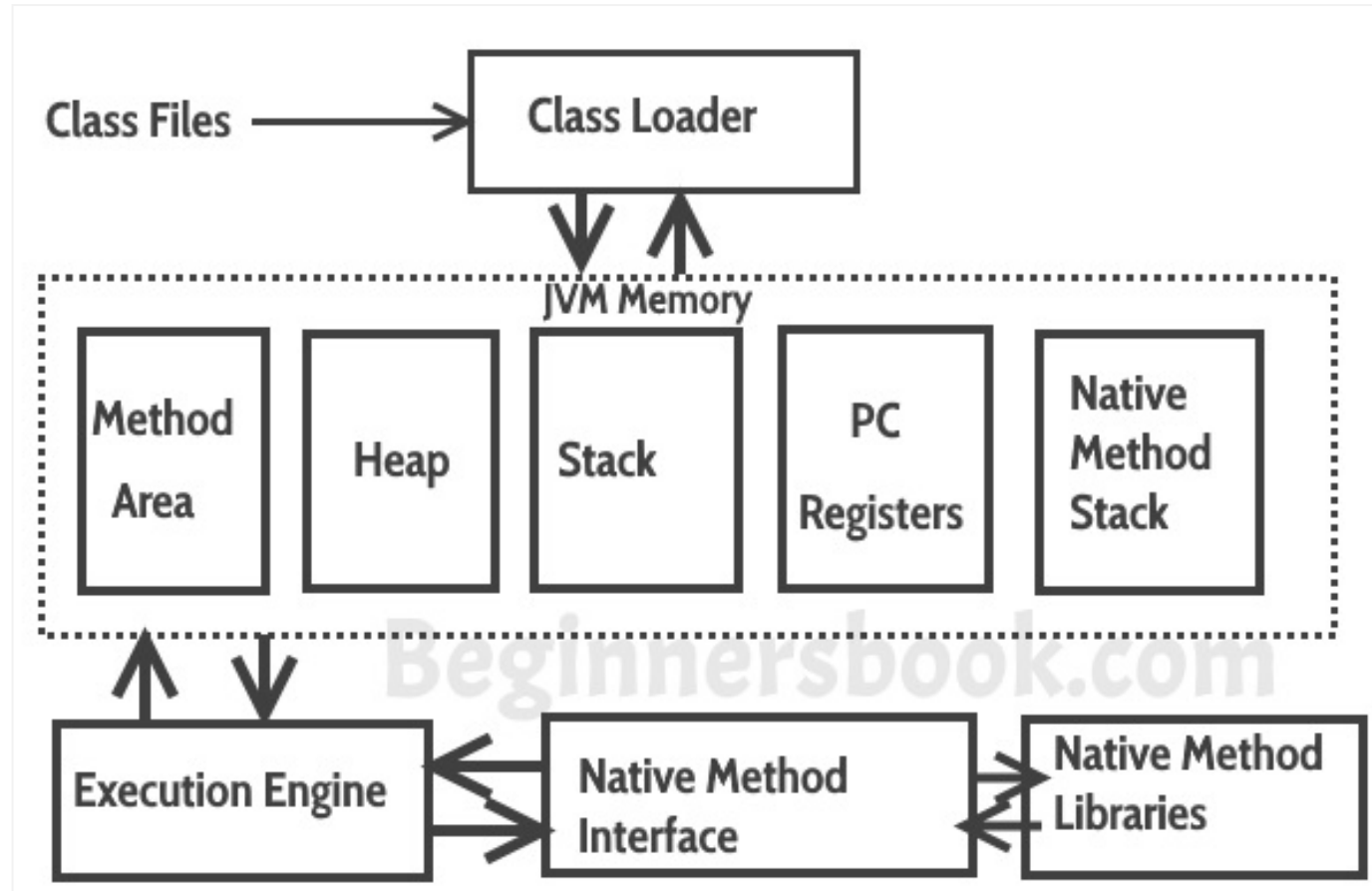


Through the Java VM, the same application is capable of running on multiple platforms.

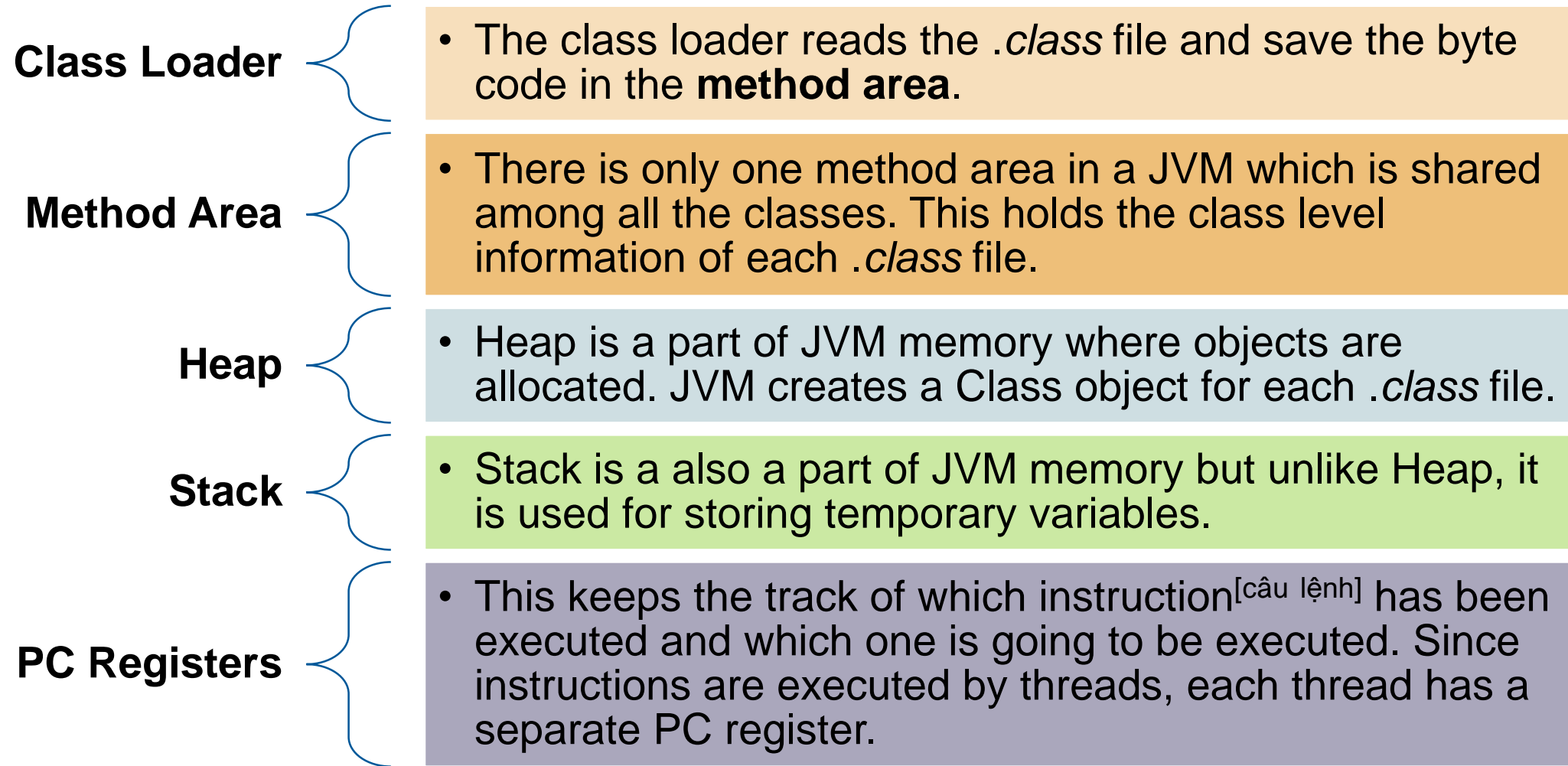
Basics of a Typical Java Environment



JVM Architecture



JVM Architecture



Native Method stack

- A native method can *access the runtime data areas of the virtual machine.*

Native Method interface

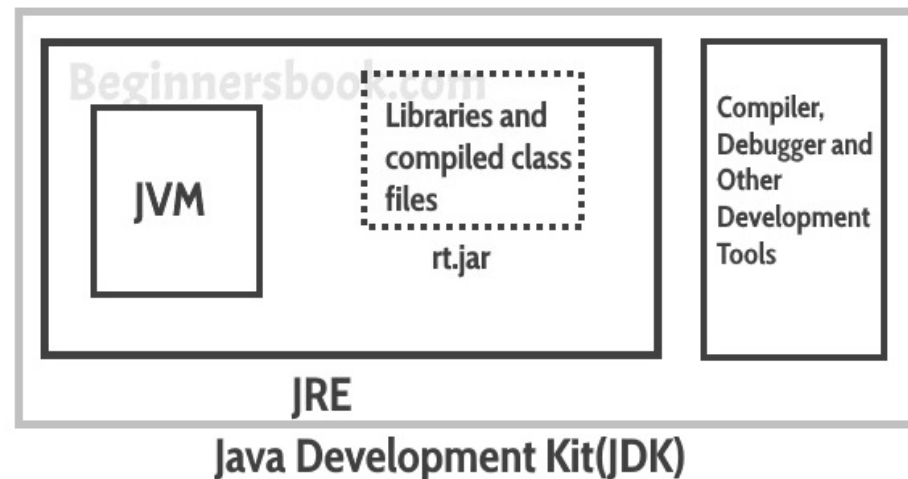
- It enables *java code to call or be called by native applications.* **Native applications** are programs that are specific to the hardware and OS of a system.

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.

▪ 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.



Section 2

First Java Program

First Sample: Printing a Line of Text

```
//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:

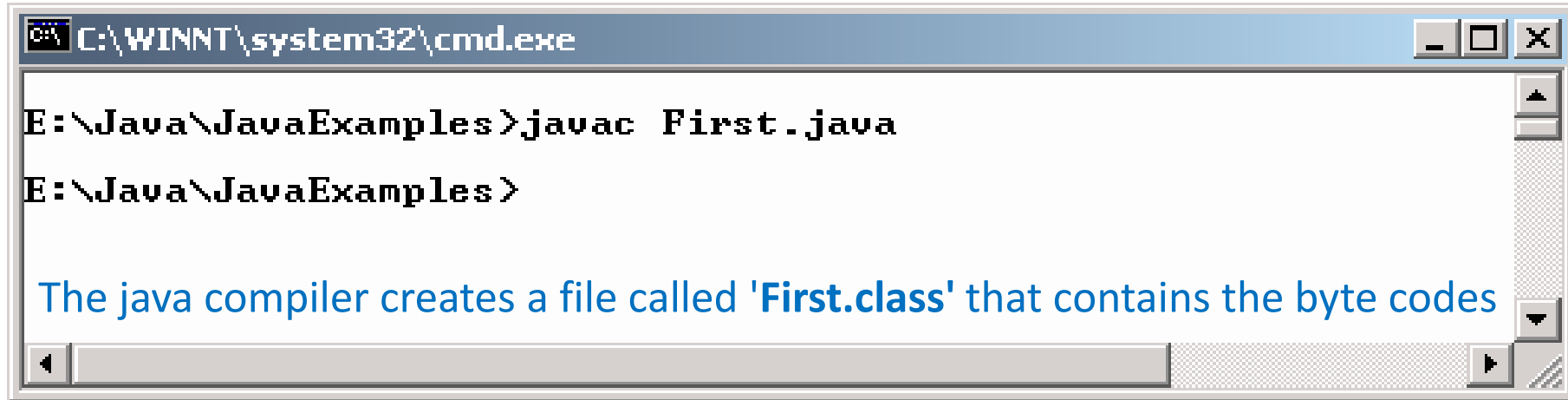
- ✓ The symbol `//` stands for commented line.
- ✓ The line `class First` declares a new class called **First**.
- ✓ `public static void main(String[] args)`

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

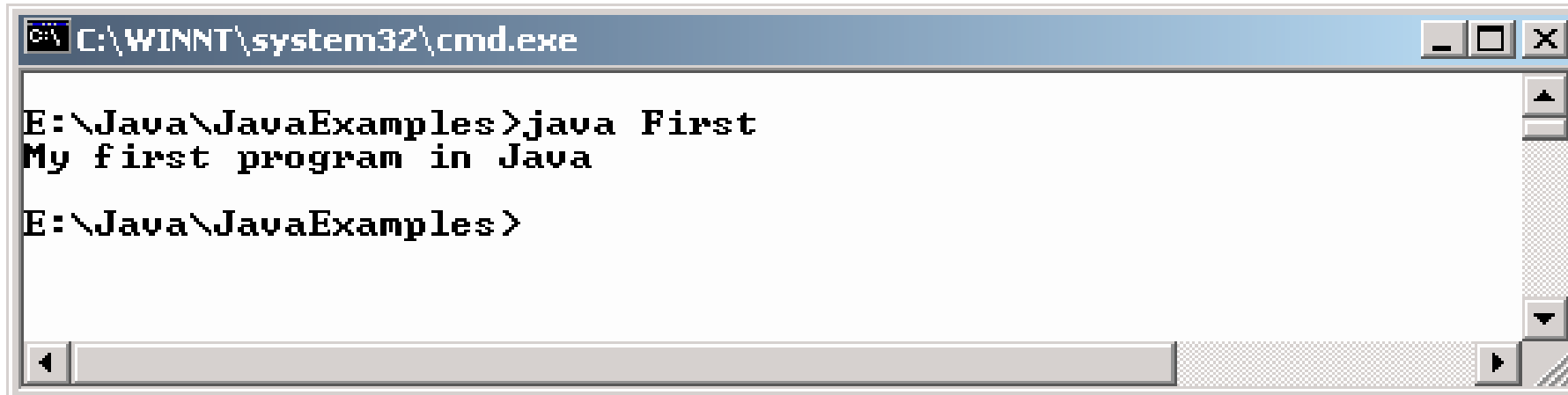


```
C:\WINNT\system32\cmd.exe

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.



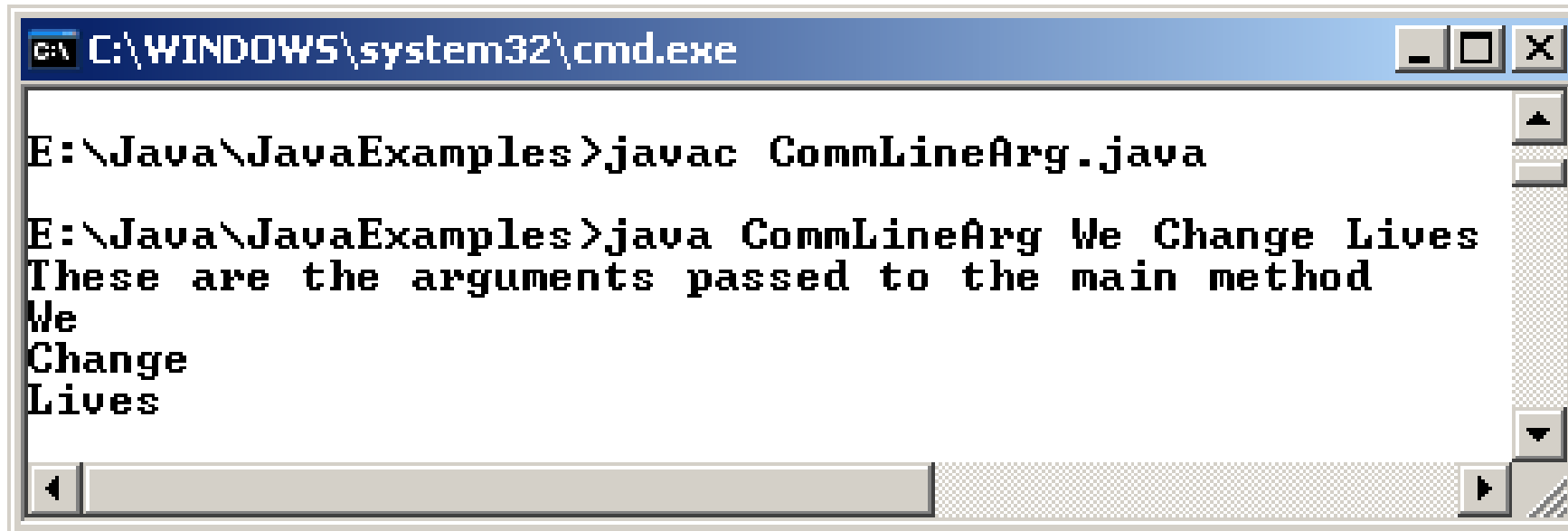
```
C:\WINNT\system32\cmd.exe

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



```
C:\WINDOWS\system32\cmd.exe

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

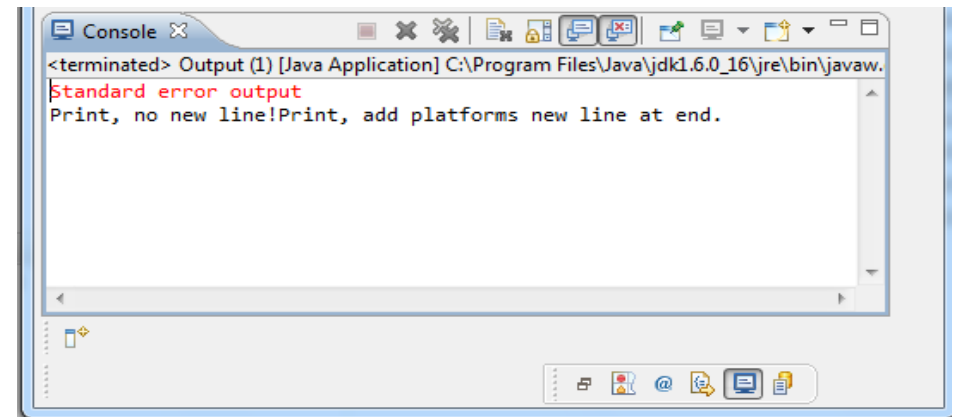
**** added in 5.0

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");  
    }  
}
```



Standard Java Input

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

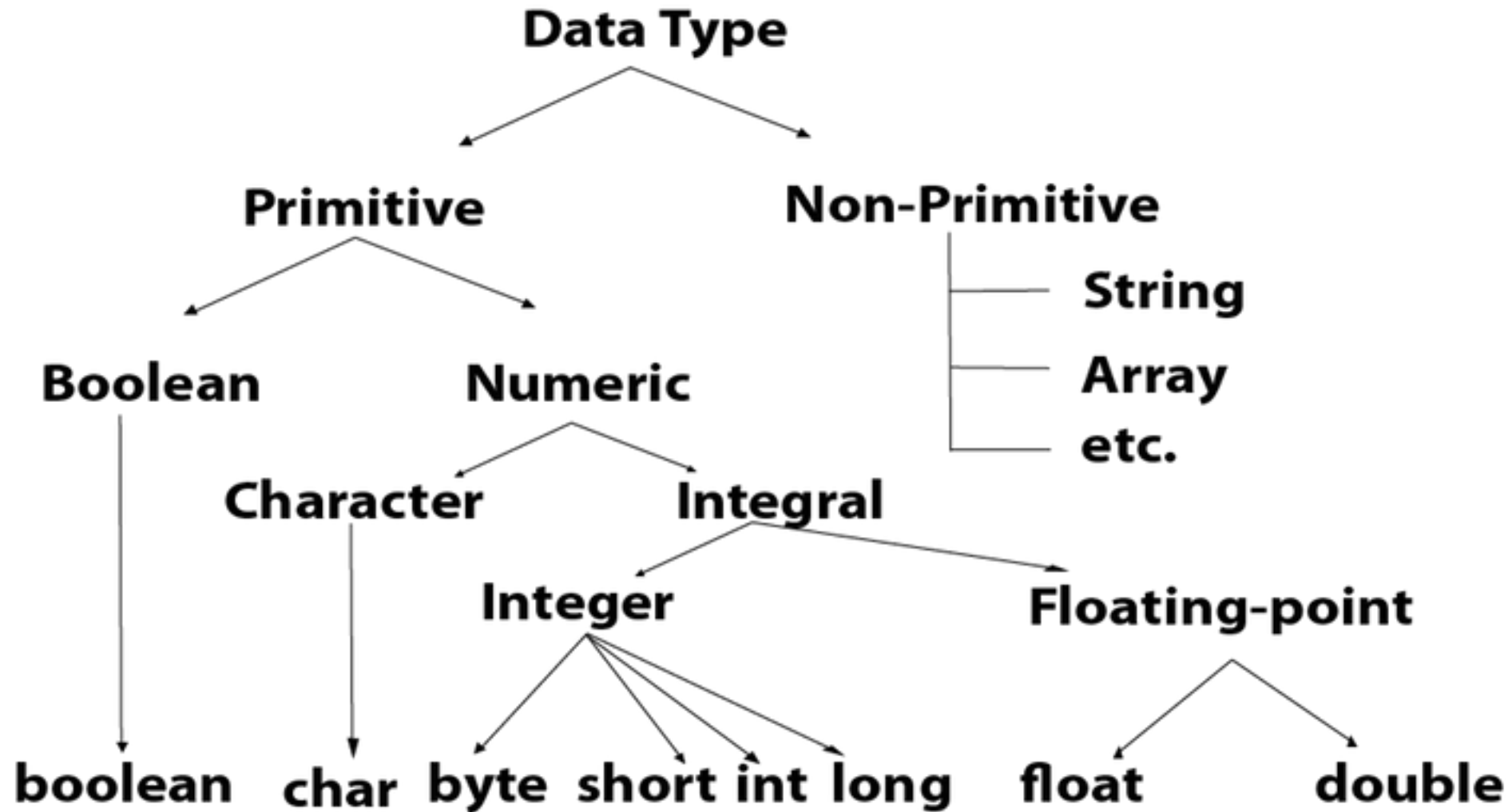
```
public class Echo {  
    public static void main(String[] args) throws IOException{  
        int ch;  
        System.out.println("Enter some text: ");  
        while ((ch = System.in.read()) != '\n') {  
            System.out.print((char) ch);  
        }  
    }  
}
```

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

```
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

```
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

`&&` Conditional-AND

`||` Conditional-OR

`?:` Ternary (shorthand for if-then-else statement)

▪ Type Comparison Operator

Instance of Compares an object to a specified type

Operators

```
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.

```
public class InstanceofOperator {  
    public static void main(String[] args) {  
        String test = "FPT";  
        boolean result;  
  
        result = test instanceof String;  
        System.out.println(result);  
    }  
}
```

Operators

```
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

▪ 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

```
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

```
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);  
    }  
}
```

Output:

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

Operators Precedence

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

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);
```

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;
```

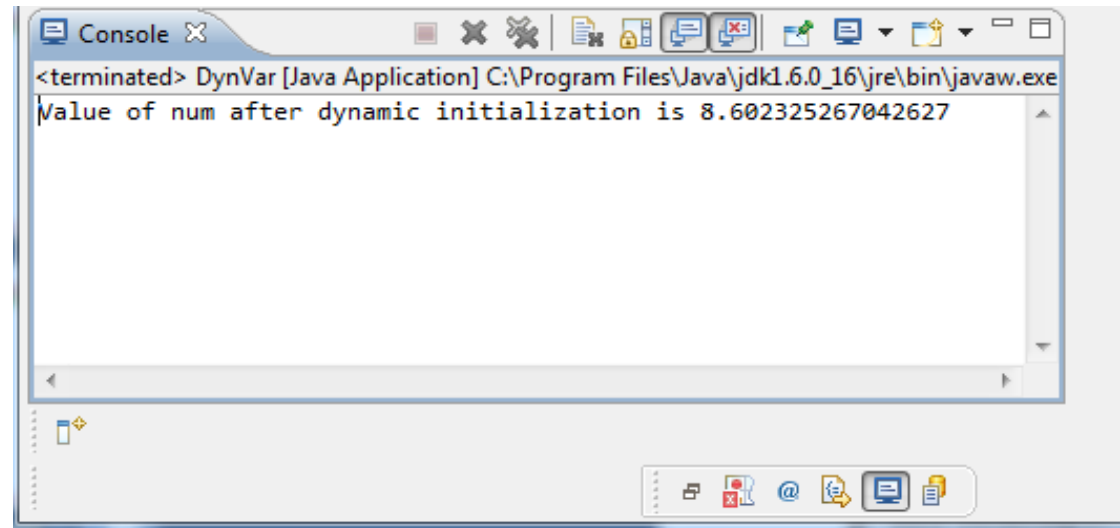
▪ Example:

```
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);  
    }  
}
```



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.

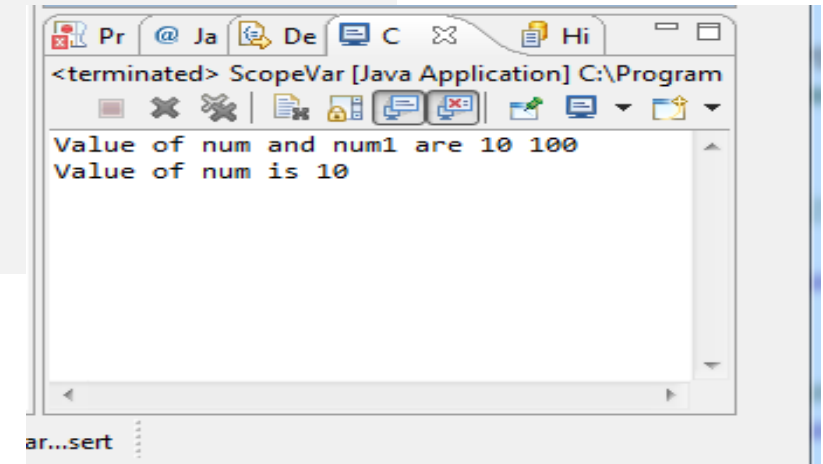


It also determines the life of an object.

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);  
    }  
}
```



SUMMARY

01. Introduction to Java

02. First Java Program

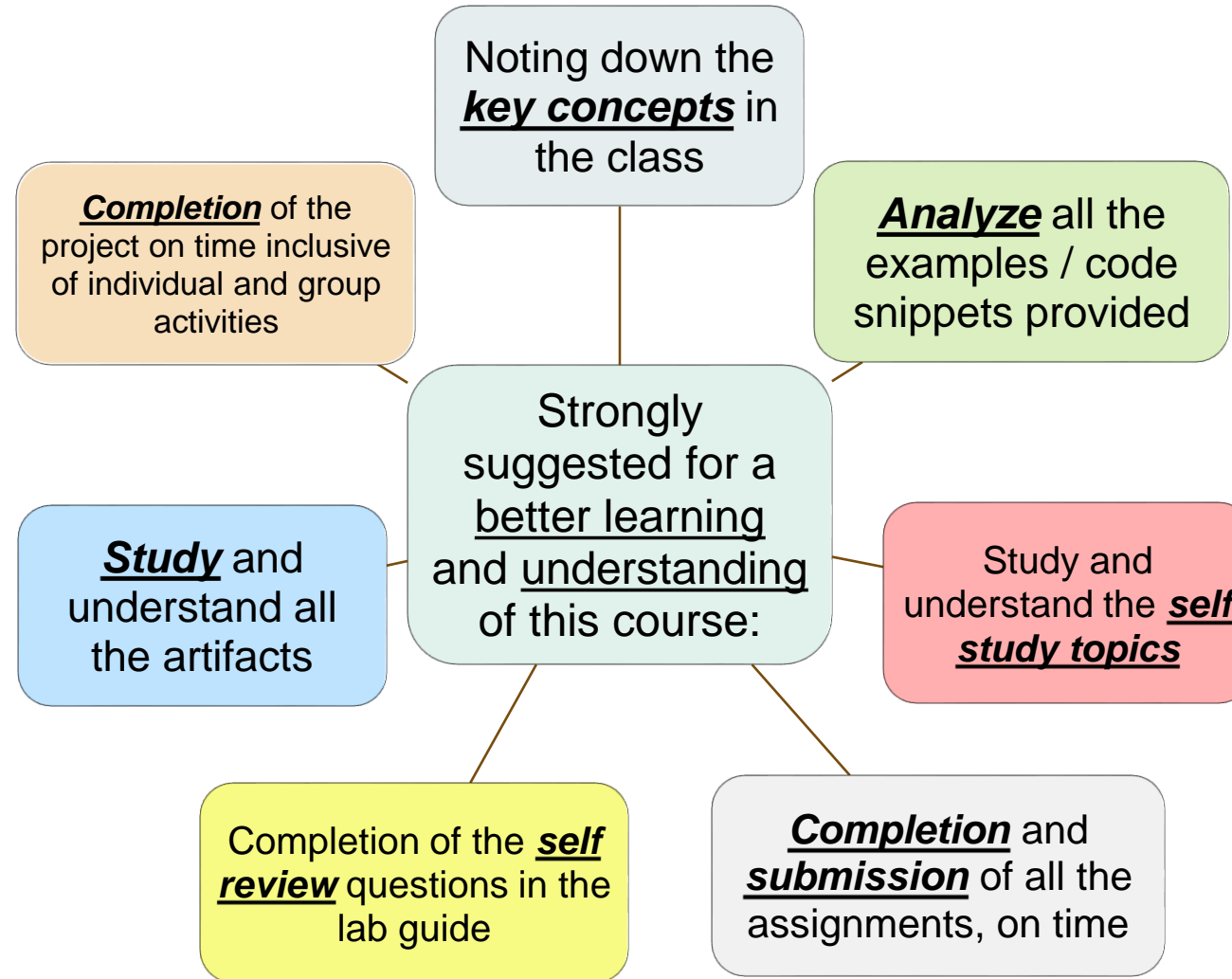
03. Basic Java Syntax

04. Java Data Types

05. Java Operators

06. Variables and Constant

Learning Approach





Questions



THANK YOU!

