CHAPTER NO:01
INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING
{ 6 MARKS}

- 1.1 Creation of java, java byte code, java characteristics
- 1.2 Abstraction, OOP Principles. -Encapsulation, Inheritance and Polymorphism
- 1.3 Constant, Variables and Data Types, Type casting
- 1.4 Operator and Expression, Arithmetic Operators, Relational Operators, Logical Operators, Assignment Operator, Increment and Decrement Operator, Bit wise Operator, Special Operator
- 1.5 Decision making with simple if, if… else, else if ladder statements, The switch statement, The conditional operator
- 1.6 Decision Making with Loops i.e. while, do and for statement, Jumps in Loops, Labelled Loops

1.1 Creation of java, java byte code, java characteristics

Java is a high-level, object-oriented programming language that was originally developed by **James Gosling** and his team at **Sun Microsystems** in the early 1990s. The language was officially released in **1995**.

- **Key Points in Java's Development:**
- **1991:** Project initiated as the "Green Project" for programming home appliances.
- **1995:** Renamed to **Java** and released to the public.
- **2009:** Sun Microsystems acquired by **Oracle Corporation**, which now maintains Java.

Design Goals of Java:

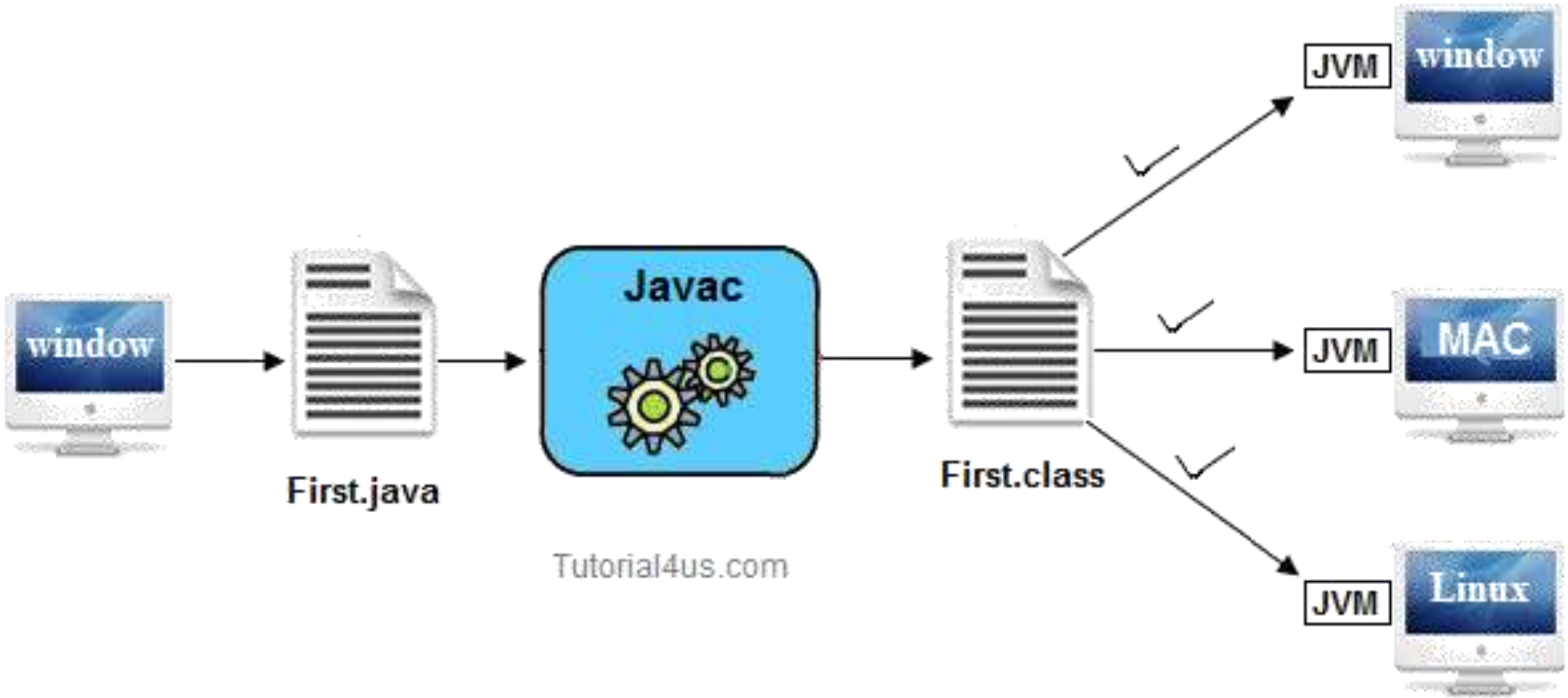
- Simple and familiar
- Object-oriented
- Platform-independent
- Secure and robust
- Architecture-Neutral:
- High performance (with Just-In-Time compilers)
- Multithreaded and dynamic

Java Byte Code:

Java programs are **not** compiled directly into machine code (like C/C++). Instead, they are compiled into an intermediate form called **Java Byte Code**.

What is Java Byte Code?

Java Byte Code is a platform-independent, low-level code generated by the Java compiler (javac) from .java source files.


window
First.java
Javac
First.class
JVM
window
JVM
MAC
JVM
Linux
Tutorial4us.com

This diagram illustrates the **platform independence of Java**:

1. A Java program (`File1.java`) is written on a Windows system.
2. It is compiled using the **Java Compiler** (`javac`) into bytecode (`File1.class`).
3. This **bytecode** is **platform-independent** and can be run on any system (Windows, Mac, Linux) that has a **Java Virtual Machine (JVM)**.
4. The **JVM on each OS** interprets the bytecode and executes it, enabling **"Write Once, Run Anywhere"** capability.

Java Program Lifecycle:

1. **Write the source code:**

```java
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

File name: `HelloWorld.java`

2. **Compile the source code:**

```bash
javac HelloWorld.java
```

This generates a file called `HelloWorld.class` containing byte code.

3. **Execute the byte code using JVM:**

```bash
java HelloWorld
```

The **JVM** loads the byte code from `HelloWorld.class`, interprets or compiles it into machine code, and runs it.

Advantages of Java Byte Code:

- **Platform Independence:** "Write once, run anywhere"—byte code can run on any device with a JVM.
- **Security:**
  - Byte code runs in a controlled environment (sandbox model / A **sandbox** is like a **safe area** where code can run, but it **can’t harm the system**.).
  - If you download a Java applet from the internet, it runs in the JVM sandbox and cannot access your files or personal data.

Performance:

- When you run Java bytecode, the **Java Virtual Machine (JVM)** doesn't just interpret it line by line slowly. Instead, it uses a special feature called the **Just-In-Time (JIT) compiler**.
- It **translates bytecode into machine code** (the language your computer's processor understands) **while the program is running**.
- This makes the program run **much faster** than interpreting bytecode step-by-step.

java characteristics :

- Simple
- Object-oriented
- Robust
- Portable
- Secure
- Multithreaded
- Architecture-Neutral:
- Interpreted & High performance
- Distributed
- Dynamic

1. Simple:

- Java is considered simple and easy to learn, especially if you already have experience with languages like C or C++.
- Java uses a clear and readable syntax, and it removes many complex features of C/C++, such as pointers and manual memory management.
- No need to worry about memory deallocation – Java handles it using automatic **garbage collection**.

- **2. Object Oriented:**
- Java is a fully object-oriented language, which means everything in Java is based on objects and classes.
- Object-oriented programming (OOP) helps you build modular, reusable, and organized code by focusing on real-world entities like objects, rather than just logic and functions.
- Key OOP Concepts in Java:
- Class – Blueprint for creating objects
- Object – Instance of a class
- Inheritance, Polymorphism, Abstraction, Encapsulation – Core principles of OOP

```
class Student
{
String name = "Akhilesh";
 int age = 35;

 void displayInfo()
    {
       System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }

}
```

```
public class StudentInfo
{
    public static void main(String[] args)
 {
        Student S1 = new Student();
        S1.displayInfo();

  }
}
```

- **3. Encapsulation:**
- Encapsulation means wrapping data and code together into a single unit (class).
- It protects data by restricting direct access using private variables and public methods.

```
class Student {
    private int age = 18;  // data is hidden and set inside the class

    public void showAge() {
        System.out.println("Age is: " + age);
    }
}

public class TestStudent {
    public static void main(String[] args) {
        Student s = new Student();
        s.showAge();  // show the age using a method
    }
}
```

- The variable age is marked as private, so it cannot be accessed directly from outside the class.
- The value 18 is already set inside the class.
- We use a public method showAge() to display the value.
- This is called Encapsulation – the data is hidden inside the class and accessed only through a method.
- It keeps the data safe and controlled.

4. Abstraction:

- Abstraction means hiding internal details and showing only essential features.
- It is implemented using abstract classes and interfaces in Java.
- In Java, you can do this using:
- **Abstract classes** (classes that can have both normal and abstract methods)
- **Interfaces** (only method definitions, no code inside)

```
abstract class Animal
 {
    // abstract method (no body) — must be implemented by subclasses
    abstract void sound();

    // normal method with body
    void sleep()
    {
        System.out.println("Animal is sleeping");
    }
}
```

```
class Dog extends Animal
{
    // provide implementation for abstract method
    void sound()
    {
        System.out.println("Dog barks");
    }
}
```

```
public class TestAbstraction
{
   public static void main(String[] args)
 {
       Dog d = new Dog();
       d.sound();  // calls Dog's version of sound()
       d.sleep();  // calls Animal's sleep() method
   }
}
```

- Animal is abstract: you cannot create an Animal object directly.
- sound() is abstract: it has no body here, but every subclass (like Dog) must write its own version.
- sleep() is a normal method that any animal can use.
- Dog implements the sound() method.
- When we create a Dog object and call sound(), we get “Dog barks” — but we don’t need to know how it works inside Animal class.

It hides complex details (you don’t see how sound() works in all animals, just that it exists).It forces all subclasses to implement certain important methods.It helps organize code in a clean, modular way.

5.Inheritance:

- Inheritance allows one class to acquire the properties and behaviors of another class.
- It promotes code **reusability** using the extends keyword.

```
class Vehicle
{
    void start()
    {
        System.out.println("Vehicle started");
    }
}
class Car extends Vehicle
 {
    void drive()
 {
        System.out.println("Car is driving");
    }
}
```

```
public class TestInheritance
 {
    public static void main(String[] args)
{
        Car c = new Car();
        c.start();   // from parent class
        c.drive();   // from child class
    }
}
```

Here, Car gets the start() method from Vehicle. No need to rewrite the same code — this is code reusability using inheritance.

6.Polymorphism:

- Polymorphism allows the same method or object to behave differently in different situations.
- It is achieved through method overloading and method overriding

OR

- Polymorphism means **one thing behaving in many ways**.
  It happens in two forms:
- **Method Overloading** (same method name, different parameters)
- **Method Overriding** (child class changes behavior of parent method)

Method Overloading Example:

```
class MathOps
{
    void add(int a, int b)
    {
        System.out.println(a + b);
    }

    void add(int a, int b, int c)
    {
        System.out.println(a + b + c);
    }
}
```

```
public class TestOverloading
{
    public static void main(String[] args)
   {
        MathOps m = new MathOps();
        m.add(2, 3);          // calls 2-arg version
        m.add(1, 2, 3);       // calls 3-arg version
    }
}
```

Method Overriding Example:

```
class Animal
{
    void sound()
   {
        System.out.println("Animal makes sound");
    }
}

class Dog extends Animal
{
    void sound()
   {
        System.out.println("Dog barks");
    }
}
```

```
public class TestPolymorphism
{
    public static void main(String[] args)
   {
        Animal a1 = new Animal();  // base class object
        Animal a2 = new Dog();     // base class reference, derived class object

        a1.sound();  // Output: Animal makes sound
        a2.sound();  // Output: Dog barks
    }
}
```

Same method name behaves differently based on situation — that's **polymorphism**.

7. Robust:

- Java is a strictly typed language, which means you have to clearly define the type of data (like int, String, etc.) when writing code.
- It also **checks your code in two steps**:
  - **At compile time** (before running) – to catch errors early
  - **At run time** (while running) – to catch any unexpected problems
- Memory management can be a difficult, tedious task in traditional programming environments.
- **For example,** in C/C++, the programmer will often manually allocate and free all dynamic memory. This sometimes leads to problems, if programmer forgets to free memory that has been previously allocated.
- Deallocation is completely automatic, because Java provides **garbage collection** for unused objects, so you don't need to free memory yourself.
- It also has **object-oriented exception handling**, which helps you manage errors in a clean and organized way.

8. Portability:

- Portability refers to the ability to run a program on different machines.
- The Java program is getting compiled into bytecode which is platform independent.
- For **example**, the same applet must can be downloaded and executed by the wide variety of CPUs, operating systems, and browsers connected to the Internet.
- There is no need to keep different versions of the applet for different computers. The same code must run on all computers.

9. Security:

- Downloading from the Internet can be risky because files might contain viruses, Trojan horses, or other harmful programs.
- Java applets run inside a secure environment called the Java execution environment.
- They are restricted and cannot access your computer’s files or disk partitions, keeping your system safe.
- Java does not allow pointers (direct memory access), which helps prevent many common programming errors and security problems.

10. Multithreaded:

- Multithreading means dividing a program into executable sub programs(threads) which can run concurrently (parallelly)
- **Examples:**
- **On a mobile home screen** : (wall paper, battery level, signal strength, location, time, etc.)
- **In a browser window: (**different tabs)
- **In a word document: (**writing process, printing process, spell check process, etc.)
- **Applications: (**Gaming, Animation, Networking programs)
- Java provides powerful and easy-to-use tools for managing multiple threads and making sure they don't interfere with each other (called synchronization). This helps programmers build smooth and interactive applications.

11. Architecture-Neutral:

- Java programs **don’t depend on the operating system**, processor, or hardware.
- This means if you upgrade your OS or change your computer’s processor, **your Java programs will still run without changes.**
- The idea behind Java and the Java Virtual Machine (JVM) is:
- **“Write once; run anywhere, any time, forever.”**

12. Distributed:

- Java is built to work well on the **Internet and networks.**
- It supports **TCP/IP protocols**, which are the basic communication rules of the internet.
- Accessing something over the Internet using a URL (like a web address) is as easy as reading a file on your computer.
- Java also supports **Remote Method Invocation (RMI),** which means a program can call methods and get results from a program running on another computer over the network.

13. Dynamic:

- Java can load classes when they are needed, not all at once.
- This is called dynamic loading.
- This makes Java programs flexible and efficient because they load only what’s necessary during execution.
- Java programs also carry extra information about their types at run-time to check and manage objects safely.
- Plus, Java can use functions written in other languages like C and C++ when needed, through something called native methods.

Writing "Hello World" Java Program

HelloWorld.java

```
class HelloWorld
{
   public static void main (String args[])
  {
    System.out.println ("Hello World");
   }
}
```

Note: Save the file with the name of class inside which main( ) resides

Java Program Structure

```
//  comments about the class
class HelloWorld
{

}
```

class header

class body

Comments can be placed almost anywhere

Java Program Structure

```
class HelloWorld
{
    //  comments about the method
    public static void main (String args[])
    {

    }

}
```

method body

method header

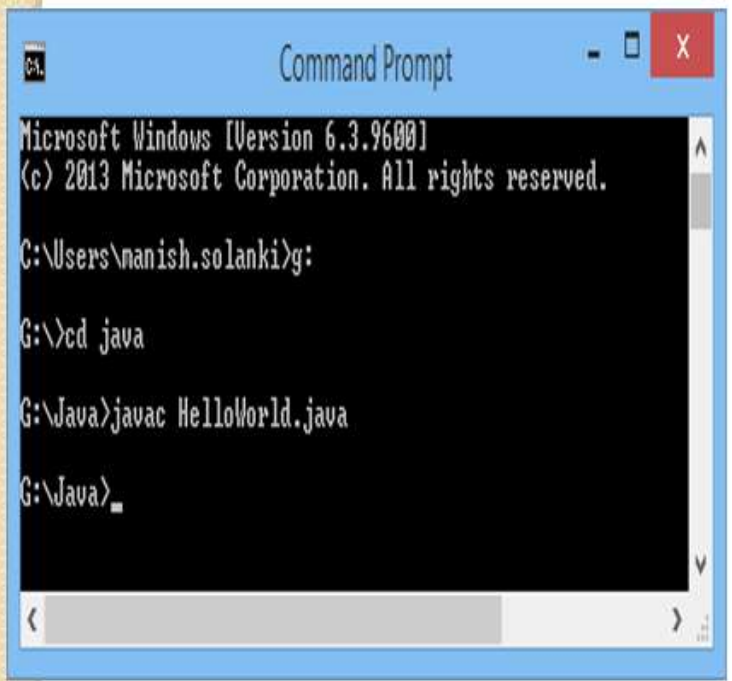
Comments

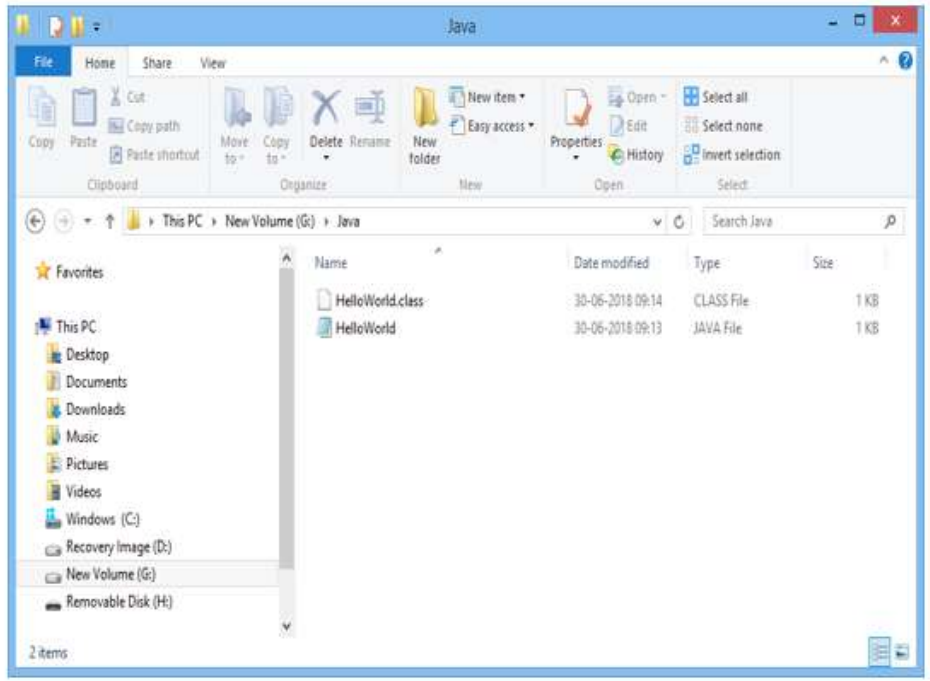
- Comments in a program are called *inline documentation*
- They should be included to explain the purpose of the program and describe processing steps
- They do not affect how a program works
- Java comments can take three forms:

```
// this comment runs to the end of the line

/*  this comment runs to the terminating
    symbol, even across line breaks       */

/** this is a javadoc comment   */
```

Compiling "Hello World" Java Program

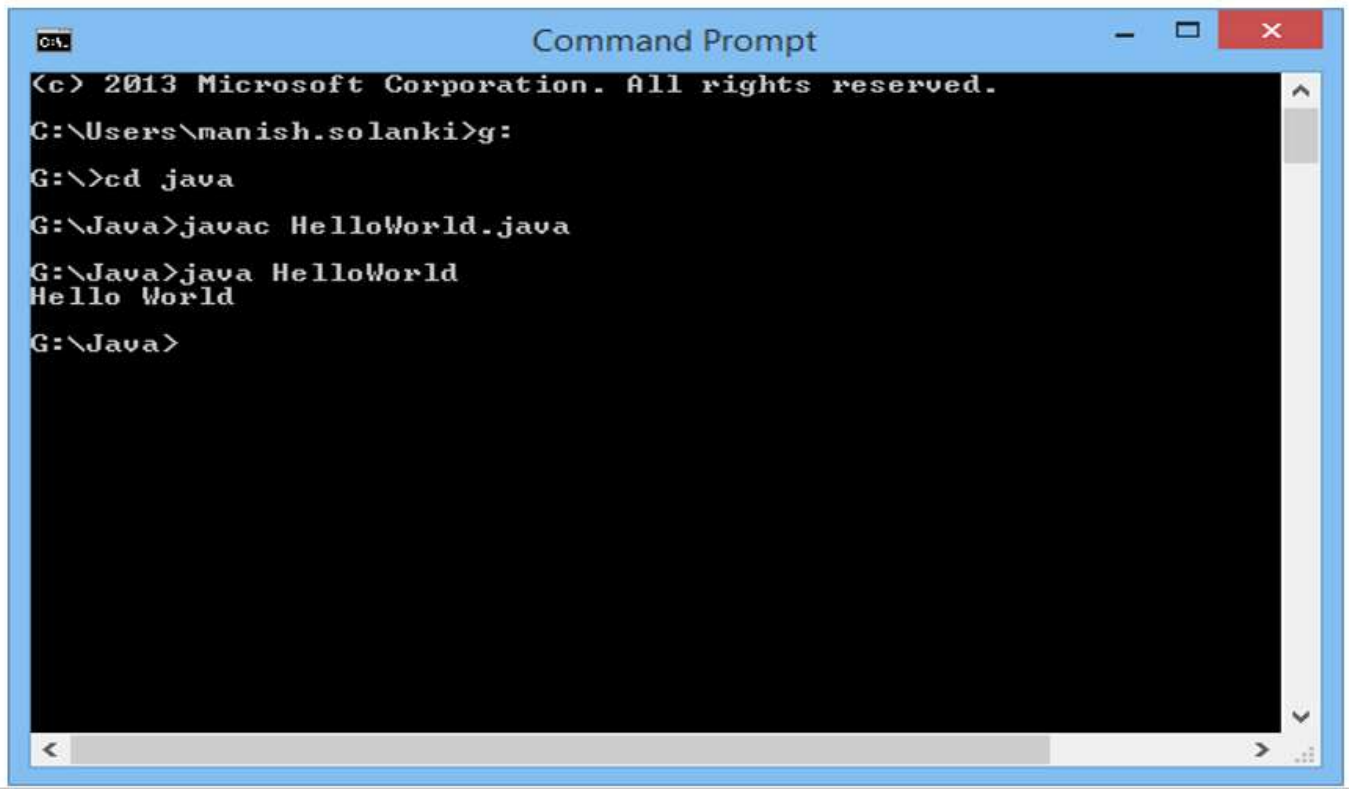
- Open Command Prompt
- Make a directory (inside which java programs are residing) as working/present directory
- Compile the program with javac program.java



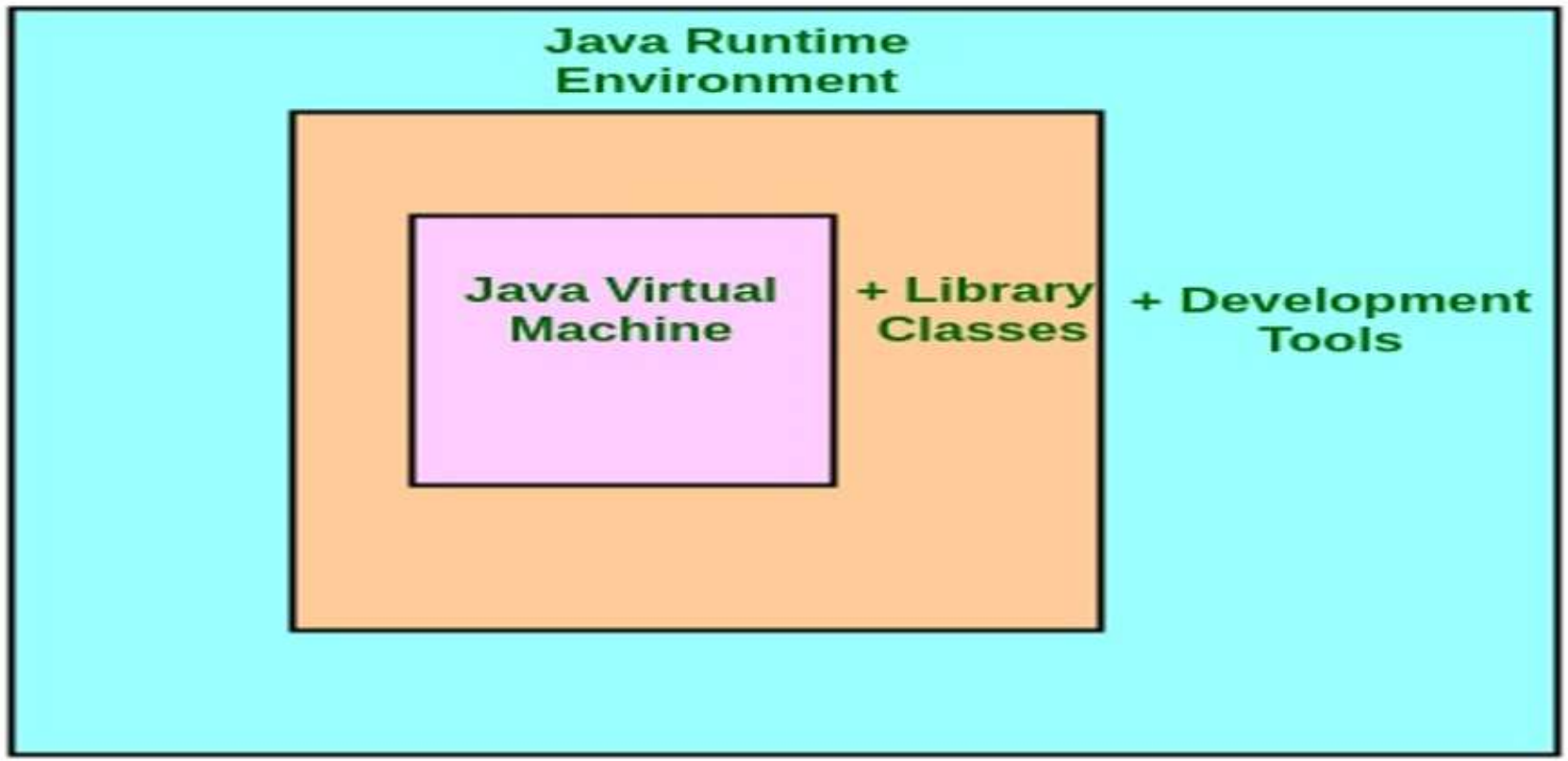


Executing "Hello World" Java Program

- After successfully compilation of program .class file (bytecode) is generated
- To execute, we have to write : java filename command



Java Ecosystem



JDK = JRE + Development Tool
JRE = JVM + Library Classes

JVM: JAVA VIRTUAL MACHINE

- JVM (Java Virtual Machine) is an abstract machine. It is called virtual machine because it doesn't physically exist.
- It is a specification that provides runtime environment in which java bytecode can be executed
- JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS are different from each other
- Functions of JVM:
    - -Loads code
    - -Verifies code
    - -Executes code
    - -Provides runtime environment

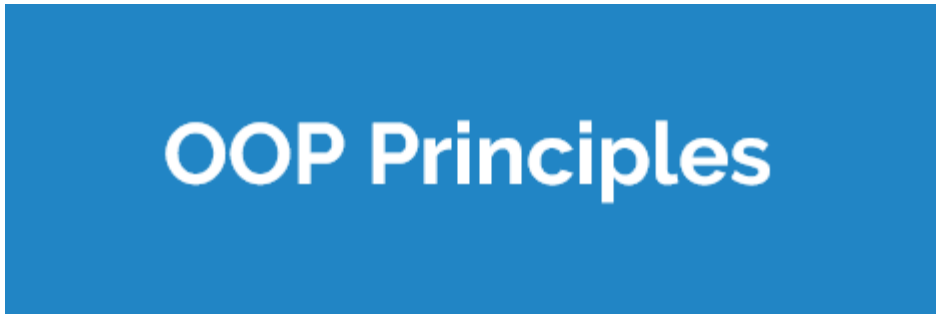
JRE: Java Runtime Environment

- JRE is an acronym for Java Runtime Environment
- It is also written as Java RTE.
- The Java Runtime Environment is a set of software tools which are used for developing java applications
- It is used to provide runtime environment. It is the implementation of JVM
- It physically exists.
- It contains set of libraries + other files that JVM uses at runtime

JDK: JAVA DEVELOPMENT KIT

- The Java Development Kit (JDK) is a software development environment which is used to develop java applications and applets.
- It physically exists.
- It contains JRE + development tools(i.e. javac,java,jar,etc.)
- JDK is an implementation of any one of the below given Java Platforms released by Oracle corporation:
- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform
- Applets(An **applet** in Java is a small program that is typically embedded in a web page and runs in a web browser using the Java Virtual Machine (JVM). Applets were commonly used in the past for adding interactive features to web applications, such as games, calculators, or visualizations.)

1.2 Abstraction, OOP Principles. -Encapsulation, Inheritance and Polymorphism



Class:

- We can create a user defined data type in Java using class.
- A class can be considered as a “Template” or “Blue Print” or “Dye” which declares data and code(function) inside it.
- A class is a logical reality from which we can create n no. of instances i.e. from a dye, we can create so many idols, from a blue print (plan/design/pattern), we can construct so many buildings physically, etc. Thus a class is a collection of objects of similar types.

Object:

- Basic run time entities in OOP i.e. a person, a bank, an employee, a student, etc.
- Are variables of type class
- Is a physical reality
- Each object contains data and code(function) to manipulate data
- Objects interact with each other by sending messages. i.e. a customer object can request an account object for the bank balance.

Data Encapsulation:

- The wrapping up of data and functions into a single unit (class) is known as Encapsulation
- Only functions declared inside the class can access the data. (outside interference is restricted – security)
- Also called as “Data Hiding” or “Information Hiding”
- The functions provide an interface between an object’s data and the program.
- Encapsulation is achieved trough access modifiers private and protected in Java.

Data Encapsulation

```
Example: class Employee
{ // data members
  private int code;
  private String name;
  private float salary;
  private int experience;
// member methods
void setEmp()
 {
    // data input
 }
 void getEmp()
 {
  // data output
 }
} // end of class
```

The attributes i.e. code, name, salary, experience are called as "data members"

The functions that operate on this data are called as "member functions " or " methods"

```
class EmployeeMain
{
   public static void main(String args[])
  {
  Employee e=new Employee();
  e.code=1; // error becuase of private access
specifier
   }
}
```

1.3 Constant, Variables and Data Types, Type casting :

- A constant is a variable whose value cannot be changed once it is assigned. In Java, we use the final keyword to declare a constant.
- **Syntax:** final dataType CONSTANT_NAME = value;
- **Example:** final double PI = 3.14159;

Variables:

- A variable is like a container that holds a value. This value can change while the program runs.
- **Syntax:** datatype variableName = value;
- **Example:** int age = 25; or String name = "John";

Data Types:

🔷 A. Primitive Data Types (8 types):

Data Type	Size	Example	Description
`int`	4 bytes	100	Whole numbers
`double`	8 bytes	3.14	Decimal numbers (more precise)
`float`	4 bytes	2.5f	Decimal numbers (less precise)
`char`	2 bytes	'A'	Single character
`boolean`	1 bit	true/false	Logical value (yes/no)
`byte`	1 byte	127	Small integers
`short`	2 bytes	32000	Shorter range integers
`long`	8 bytes	1000000000L	Very large integers

🔷 B. Non-Primitive Data Types:

Includes **String**, **Arrays**, **Objects**, etc.

```java
String city = "London";
int[] marks = {90, 80, 70}; // Array of integers
```

Type casting : Type Casting means converting one data type into another.

A. Implicit Casting (Widening)

- Java **automatically converts** smaller data types into larger ones (no data loss).

```
int a = 10;
double b = a; // int is automatically converted to double
```

B. Explicit Casting (Narrowing):

- You have to **manually convert** larger data types into smaller ones (possible data loss).

```
double x = 9.78;
int y = (int) x; // double to int (Result: 9 – decimal part is lost)
```

EXAMPLE:

```
public class Example {
    public static void main(String[] args) {
        // Variables
        int age = 20;
        String name = "Alice";

        // Constant
        final double PI = 3.14159;
```

```java
// Data Types
boolean isPassed = true;
char grade = 'A';
float percentage = 85.6f;

// Type Casting
int a = 50;
double b = a; // Implicit casting
double price = 99.99;
int newPrice = (int) price; // Explicit casting
```

```
        // Output
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Grade: " + grade);
        System.out.println("Passed: " + isPassed);
        System.out.println("PI: " + PI);
        System.out.println("Implicit Casting (int to double): " + b);
        System.out.println("Explicit Casting (double to int): " + newPrice);
    }
}
```

1.4 Operator and Expression, Arithmetic Operators, Relational Operators, Logical Operators, Assignment Operator, Increment and Decrement Operator, Bit wise Operator, Special Operator .

Bit wise Operator:

Bitwise operators work on **individual bits (0 or 1)** of integers. These are mainly used for low-level programming, performance, or bit manipulation.

📋 List of Bitwise Operators:

Operator	Name	Description	Example (`a=5, b=3`)
`&`	AND	1 if **both** bits are 1	`a & b = 1`
`	`	OR	1 if **any one** of the bits is 1
`^`	XOR	1 if bits are **different**	`a ^ b = 6`
`~`	NOT	Inverts each bit (1→0, 0→1)	`~a = -6`
`<<`	Left Shift	Shifts bits to the left, adds 0s on the right	`a << 1 = 10`
`>>`	Right Shift	Shifts bits to the right, keeps the sign bit	`a >> 1 = 2`
`>>>`	Unsigned Right Shift	Same as right shift, but fills 0s from the left	`a >>> 1 = 2`

```java
public class BitwiseExample {
    public static void main(String[] args) {
        int a = 5;  // Binary: 0101
        int b = 3;  // Binary: 0011

        System.out.println("a & b: " + (a & b));    // 1
        System.out.println("a | b: " + (a | b));    // 7
        System.out.println("a ^ b: " + (a ^ b));    // 6
        System.out.println("~a: " + (~a));          // -6
        System.out.println("a << 1: " + (a << 1)); // 10
        System.out.println("a >> 1: " + (a >> 1)); // 2
    }
}
```

Special Operator: Java has some **special-purpose operators** that don't fit into other categories like arithmetic or logical.

📋 Common Special Operators:

Operator	Name	Description	Example
`instanceof`	Instance Check	Checks if an object belongs to a class	`obj instanceof String`
`?:`	Ternary Operator	Short form of if-else condition	`a > b ? a : b`
`this`	Current Object Ref	Refers to the current object in a method	`this.name = name;`
`super`	Parent Class Ref	Refers to parent class constructor or method	`super.methodName();`

1. `instanceof`

```java
String s = "Hello";
System.out.println(s instanceof String);  // true
```

2. `?:` (Ternary)

```java
int a = 10, b = 20;
int max = (a > b) ? a : b;  // returns 20
System.out.println("Max: " + max);
```

Syntax:

```
condition ? value_if_true : value_if_false;
```

```
int num = 7;
String result = (num % 2 == 0) ? "Even" : "Odd";
System.out.println("Number is: " + result);
```

Part	Description
`condition`	Expression to check (like `a > b`)
`?`	Separator between condition and true value
`value_if_true`	Returned if condition is true
`:`	Separator between true and false values
`value_if_false`	Returned if condition is false

3. `this`

```java
class Student {
    String name;
    Student(String name) {
        this.name = name;  // refers to current object's name
    }
}
```

The **this** keyword refers to the **current object** — the object whose method or constructor is being called.

We mostly use this to **avoid confusion** when local (method/constructor) variables have **the same name** as instance variables (class variables).

```java
class Student {
    String name;

    // Constructor
    Student(String name) {
        this.name = name;  // 'this.name' = class variable, 'name' = constructor
    }

    void display() {
        System.out.println("Name: " + this.name); // optional use of this
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student("Ravi");
        s1.display();  // Output: Name: Ravi
    }
}
```

4. `super`

```java
class Animal {
    void sound() {
        System.out.println("Animal sound");
    }
}

class Dog extends Animal {
    void sound() {
        super.sound();  // calls parent class method
        System.out.println("Dog barks");
    }
}
```

1.5 Decision making with simple if, if… else, else if ladder statements, The switch statement, The conditional operator :

The conditional operator :

```
int marks = 45;
String status = (marks >= 50) ? "Pass" : "Fail";
System.out.println("Result: " + status);
```

1.6 Decision Making with Loops i.e. while, do and for statement, Jumps in Loops, Labelled Loops :

🔶 1. **Jumps in Loops**

In Java, jump statements are used to control the **flow of loops**. The main jump statements are:

✅ A. `break`

> Used to **exit** the loop immediately.

java Copy Edit

```java
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        break;
    }
    System.out.println(i);
}
```

🖥️ Output:

Copy Edit

```
1
2
```

👉 Loop stops when `i == 3`.

✅ B. `continue`

> Skips the **current iteration** and goes to the **next** one.

java

Copy Edit

```java
for (int i = 1; i <= 5; i++) {
    if (i == 3) {
        continue;
    }
    System.out.println(i);
}
```

💻 Output:

Copy Edit

```
1
2
4
5
```

👉 `i == 3` is skipped.

✅ C. `return`

> Exits the **method completely**, even if inside a loop.

java

Copy Edit

```java
public class Main {
    public static void main(String[] args) {
        for (int i = 1; i <= 5; i++) {
            if (i == 3) {
                return;
            }
            System.out.println(i);
        }
        // This line won't run
        System.out.println("End of loop");
    }
}
```

🖥️ Output:

Copy Edit

```
1
2
```

👉 The method exits completely when `i == 3`.

🔶 2. **Labelled Loops**

✅ What are Labelled Loops?

Java allows you to give a **name (label)** to a loop. This is useful when you have **nested loops**, and you want to break or continue **outer** loops, not just the inner one.

🔷 Syntax:

java Copy Edit

```java
labelName:
for (...) {
    for (...) {
        break labelName;
    }
}
```

✅ Example: `break` with label

java

Copy Edit

```java
outer:
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        if (i == 2 && j == 2) {
            break outer; // exits the outer loop
        }
        System.out.println(i + " " + j);
    }
}
```

💻 Output:

Copy Edit

```
1 1
1 2
1 3
2 1
```

👉 When `i == 2` and `j == 2`, it breaks out of the outer loop.

✅ Example: `continue` with label

java

Copy Edit

```java
outer:
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 3; j++) {
        if (j == 2) {
            continue outer; // skips to next iteration of outer loop
        }
        System.out.println(i + " " + j);
    }
}
```

🖥️ Output:

Copy Edit

```
1 1
2 1
3 1
```

👉 When `j == 2`, it skips the remaining part of the **outer** loop and continues with the next `i`.

Basic Building Blocks

- **Literals (Constants)**
  - ✓ **integer i.e. 123, -23, 10101, etc.**
  - ✓ **floating point: 2.546, 0.00023, etc.**
  - ✓ **Character : 'A', '#', etc.**
  - ✓ **String: "Hello" , "Java" ,etc.**
- **Variables : Rules are same as C/C++ (except $ sign is allowed in Java)**
- **Keywords : Reserved words i.e. class, float, if, for, etc.**

Java Keywords

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

Type Conversion

Type Conversion and Casting

Java's Automatic Conversions: (Implicit Type Casting)

- ✓ **When one type of data is assigned to another type of variable, an *automatic type conversion* will take place if the following two conditions are met:**

1. **The two types are compatible.**
2. **The destination type is larger than the source type.**

Type Compatibility:

- ✓ The numeric types, including integer and floating-point types, are compatible with each other

Examples:

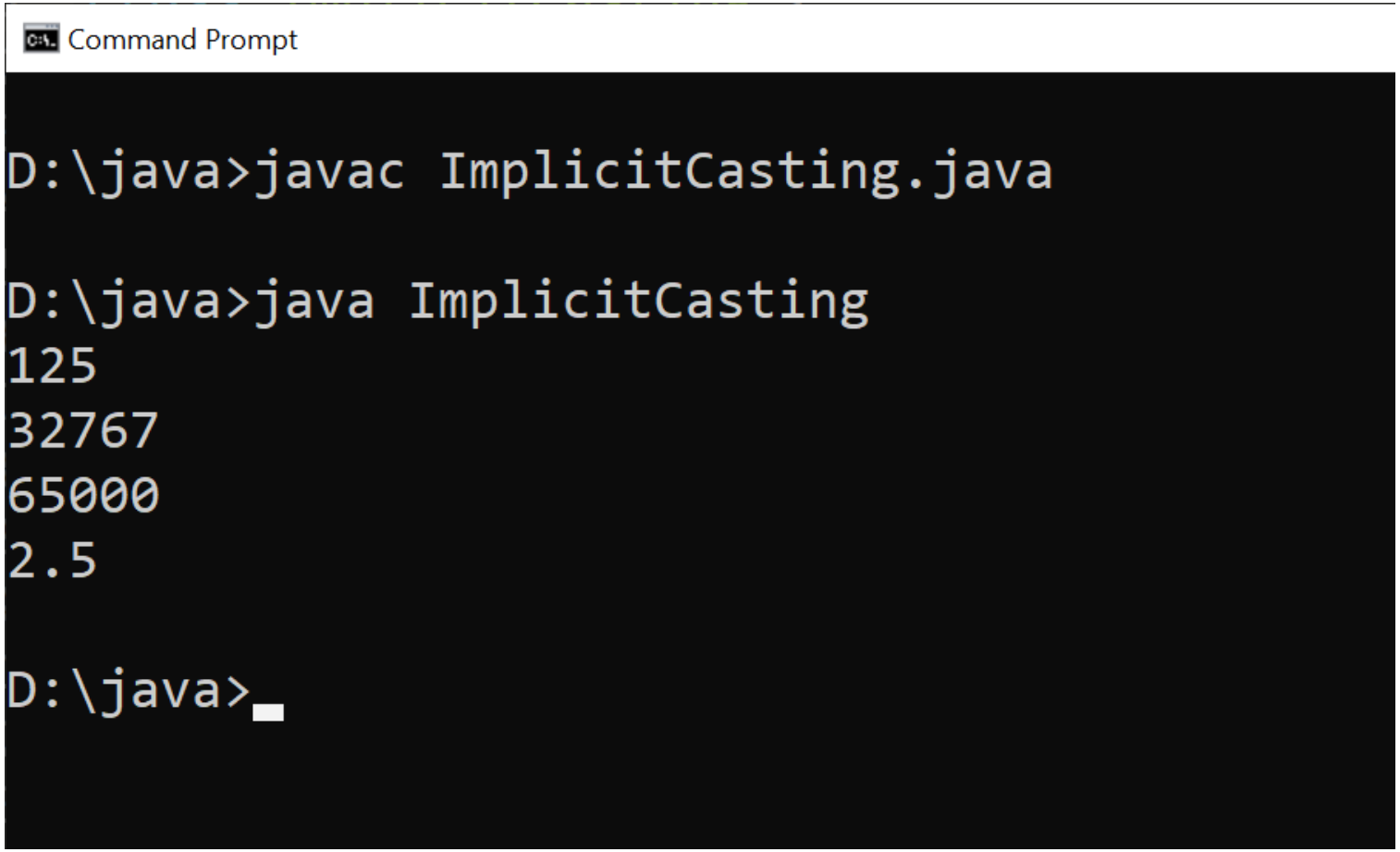
I. byte type data can be assigned to int, short and long
II. short type data can be assigned to int and long
III. int type data can be assigned to long and float
IV. float type data can be assigned to double

✓No automatic conversions from the numeric types to char or boolean.

✓char and boolean are not compatible with each other.

```java
class ImplicitCasting {
public static void main(String args[]) {
long l;
byte b = 125;
short s = 32767;
int i= 65000;
l=b;
System.out.println(l);
l=s;
System.out.println(l);
l=i;
System.out.println(l);
float f=2.5f;
double d=f;
System.out.println(d);
}
}
```

Type Conversion and Casting



Type Conversion and Casting

Casting Incompatible Types:(Explicit Type Casting)

- ✓**What if you want to assign an int value to a byte variable?**
- ✓**This conversion will not** be performed automatically, because a **byte is smaller than an int (violation of rule 2)**

- ✓**This kind of conversion is sometimes called a *narrowing conversion*, *since you are explicitly making the value narrower* so that it can fit into the target type**

Type Conversion and Casting

Casting Incompatible Types:(Explicit Type Casting)

✓To create a conversion between two incompatible types, you must use a cast.

✓A *cast is* simply an explicit type conversion.

(target-type) value

Here, *target-type = type to which value will be converted*

Example: int a=23;

byte b;

b= (byte) a;

Type Conversion and Casting

Casting Incompatible Types:

```
1  class Conversion {
2  public static void main(String args[]) {
3  byte b;
4  int i = 257;
5  double d = 323.142;
6  System.out.println("Conversion of int to byte.");
7  b = (byte) i;
8  System.out.println("i and b " + i + " " + b);
9  System.out.println("Conversion of double to int.");
10 i = (int) d;
11 System.out.println("d and i " + d + " " + i);
12 System.out.println("Conversion of double to byte.");
```

Casting Incompatible Types:

- **How int to byte conversion would take place?**
- **Answer#1:**

➢ **byte requires 8-bits (1 byte) storage.**

➢ **byte can store : -128 to +127 values only.**

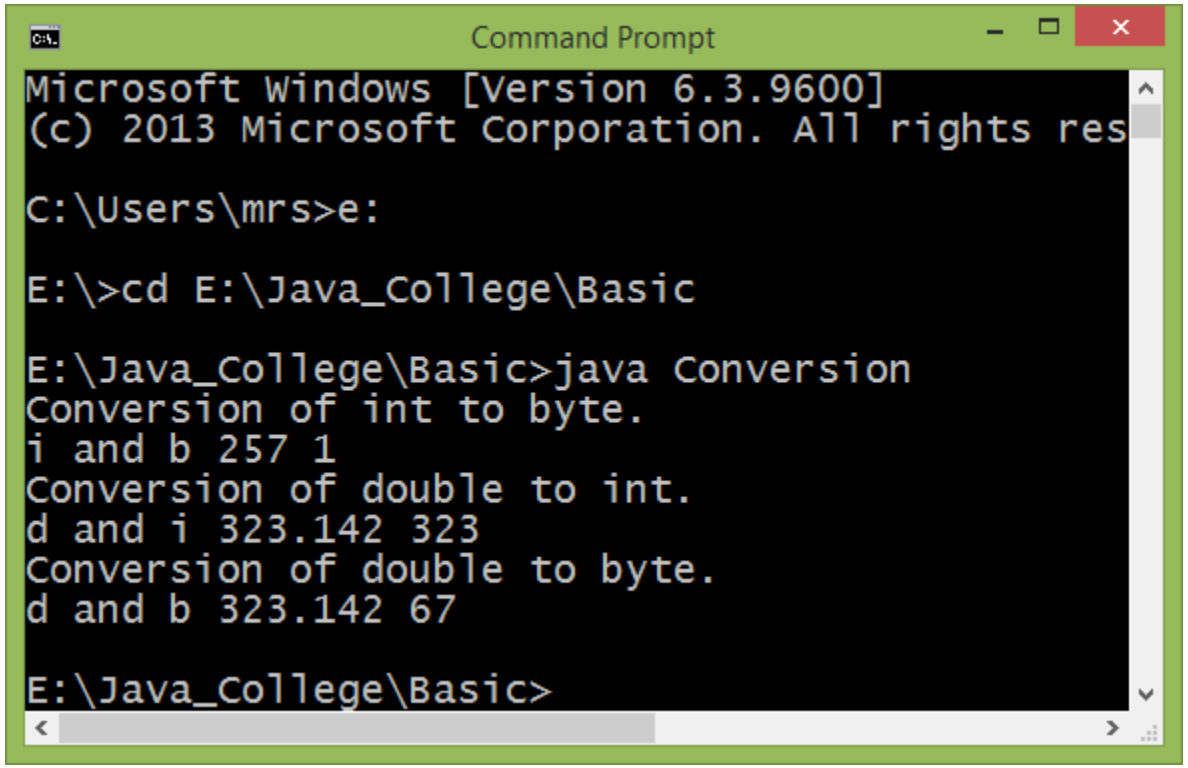
- **While assigning int value beyond the range of target type i.e. byte:**
- **result =value%2^n**
-
- n= no. of bits required for destination data type
-
- **result=257%256 (2^8 = 256)**
- **= 1**

- **How int to byte conversion would take place?**
- Answer#2:
- **byte requires 8-bits (1 byte) storage.**
- **byte can store : -128 to +127 values only.**

- Represent 257 into binary form:

							256	128	64	32	16	8	4	2	1
							1	0	0	0	0	0	0	0	1

As byte can store only 8-bits the result will be 1


Command Prompt
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights res
C:\Users\mrs>e:
E:\>cd E:\Java_College\Basic
E:\Java_College\Basic>java Conversion
Conversion of int to byte.
i and b 257 1
Conversion of double to int.
d and i 323.142 323
Conversion of double to byte.
d and b 323.142 67
E:\Java_College\Basic>

1.4 Operator and Expression, Arithmetic Operators, Relational Operators, Logical Operators, Assignment Operator, Increment and Decrement Operator, Bit wise Operator, Special Operator

1.5 Decision making with simple if, if… else, else if ladder statements, The switch statement, The conditional operator

1.6 Decision Making with Loops i.e. while, do and for statement, Jumps in Loops, Labelled Loops