JAVA

BASICS OF JAVA

- What is JAVA ?
- Where JAVA is used ?
- What type of applications are created using JAVA ?
- Why use JAVA ?

What is JAVA ?

- JAVA is a programming language and a platform

What is Programming Language ?

- A programming language is a formal computer language designed to communicate instructions to a machine, particularly a computer.
- Programming languages can used to create programs to control the behaviour of machine or to express algorithms.

What is Programming Language ?

- The most basic (called low level) computer language is the machine language that uses binary(1’s and 0’s) code which is computer can run very fast without using translator (or) interpreter. But it is tedious and complex.

What is Programming Language ?

- The high - level languages (such as C, java) are much simpler to use but need to use another program to convert the high - level code into the machine code, therefore slower.

Example

Program to ADD 1 and 2

Assembly Language

- Store 1 at memory location say A
- Store 2 at memory location say B
- ADD contents of location A and B
- Store RESULT

Store 1 at memory location say A
Store 2 at memory location say B ADD contents of location A and B Store RESULT

Assembler

10100010

C - Language
printf(1+2)

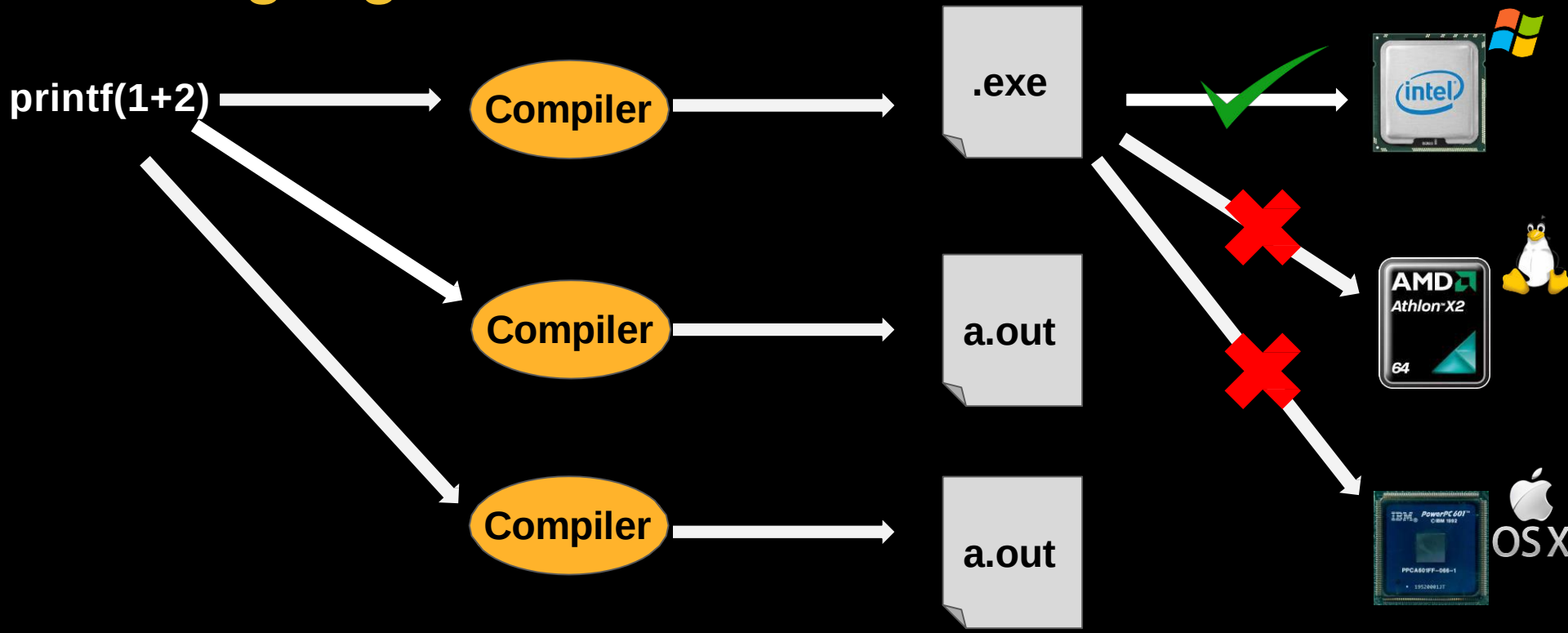
Compiler

Store 1 at memory location say A
Store 2 at memory location say B ADD contents of location A and B Store RESULT

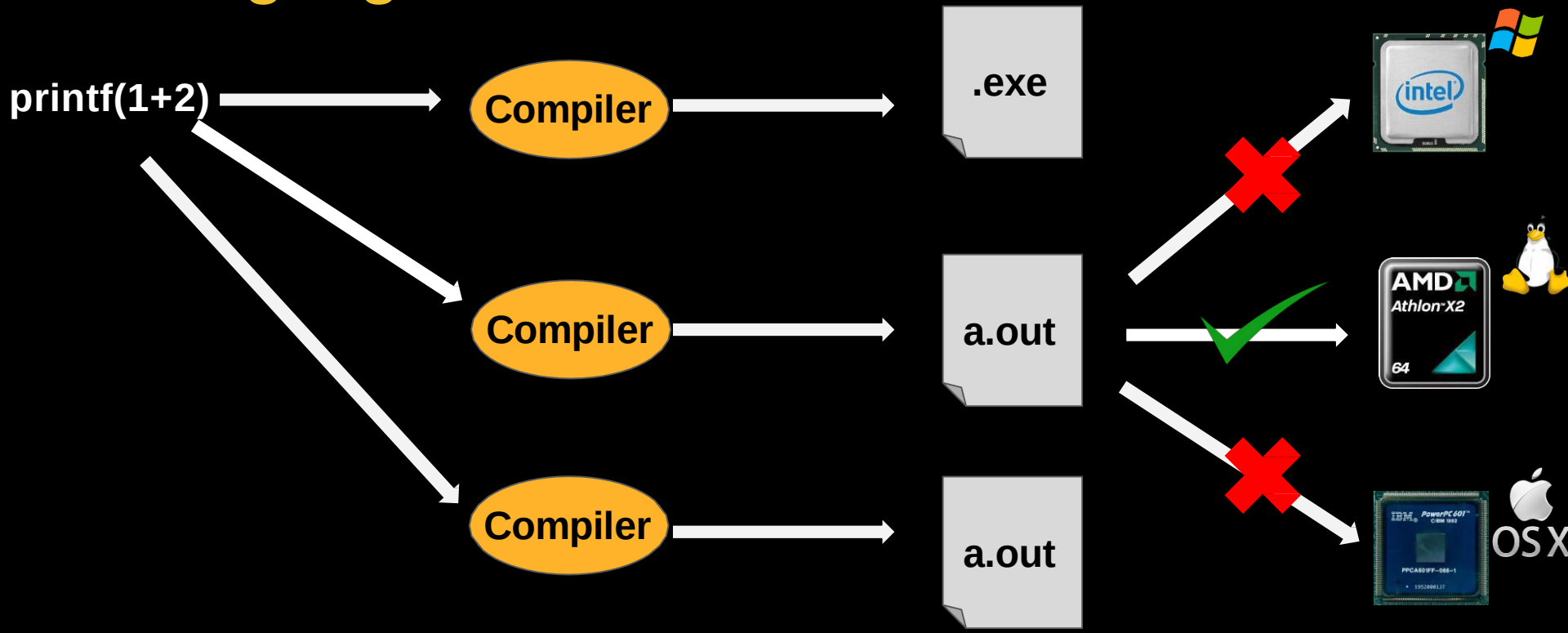
Assembler

10100010

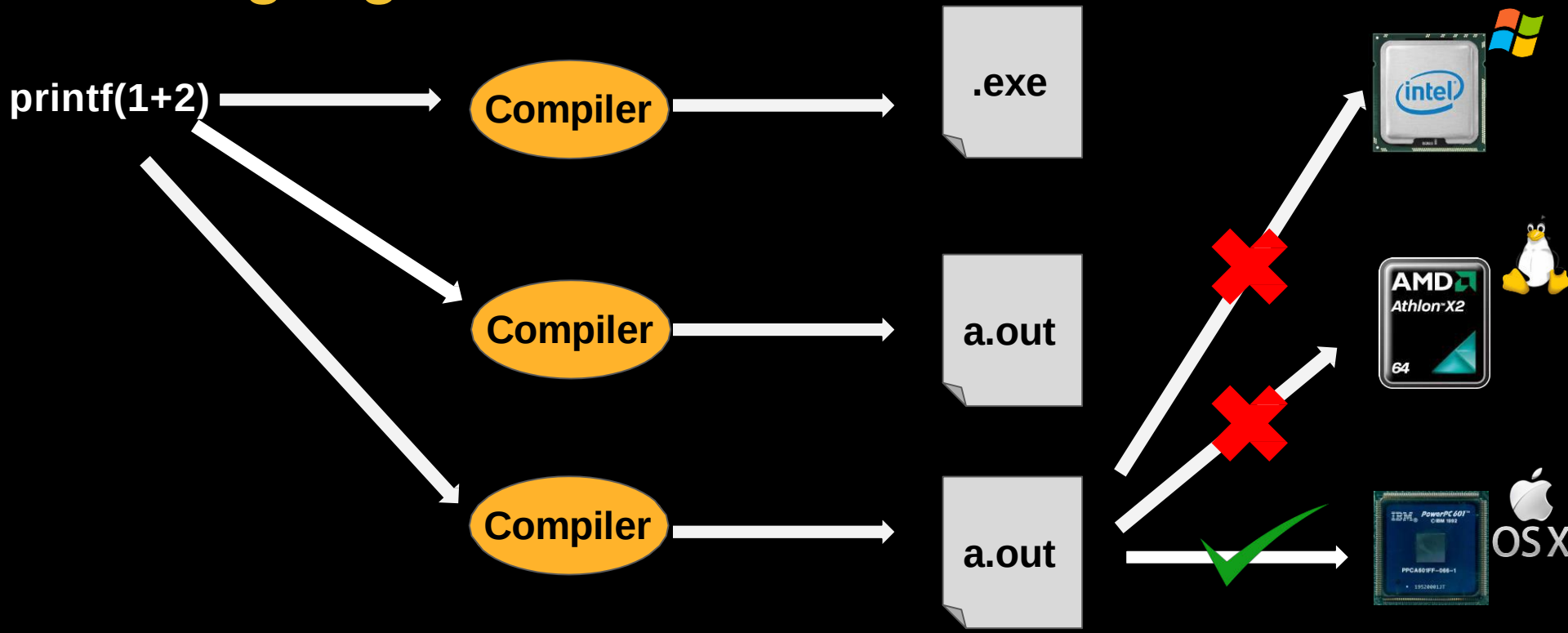
C - Language


printf(1+2)
Compiler
Compiler
Compiler
.exe
a.out
a.out
intel
AMD
Athlon X2
64
IBM
PowerPC 601
OS X

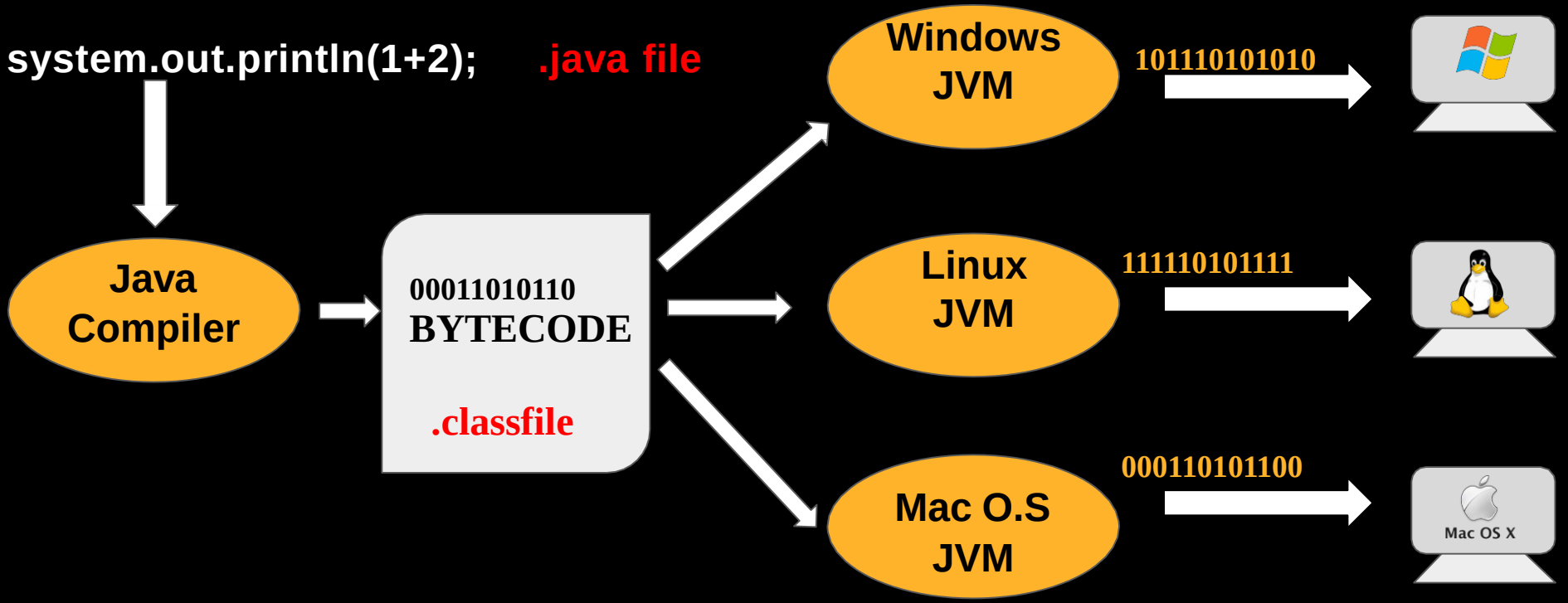
C - Language


printf(1+2)
Compiler
Compiler
Compiler
.exe
a.out
a.out
intel
AMD
Athlon X2
64
IBM
PowerPC 601
OS X

C - Language

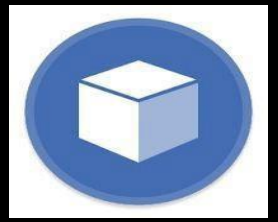

printf(1+2)
Compiler
Compiler
Compiler
.exe
a.out
a.out
intel
AMD
Athlon X2
64
IBM
PowerPC 601
OS X

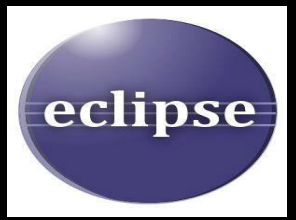
Java - Language



Where java is used

- According to SUN, 3 billion devices run java. There are many devices where java is currently used. Some of them are..
- Desktop Applications:





Where java is used?

- **Web Applications:**

- **Games:**

Types of Java Applications

- **Standalone application**: It is also known as desktop application (or) window-based application.

- Standalone software is a computer software that can work offline i.e it doesn’t necessarily require network connection.

Types of Java Applications

- **Web application**: An application that is stored on a remote server and delivered over the internet through a browser interface.
- **Enterprise application:** Enterprise software is computer software used to satisfy the needs of an organization rather than individual users. Such organizations would include business, schools, clubs, charities or governments.

Types of Java Applications

- **Mobile Application**: A mobile application, most commonly referred to as an app, is a type of application software designed to run on a mobile device, such as smartphone or tablet computer.
  - Mobile applications frequently serve to provide users with similar services to those accessed on PCs.
  - Apps are generally small, individual software units with limited function.

History of Java

- The history of java starts from Green Team. Java Team members (also known as Green Team). Initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.

- James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.

History of Java

- Firstly, it was called “Greentalk” by james Gosling and file extension was .gt.
- After that, it was called Oak and was developed as part of the Green Project
- Why Oak?

  Name is inspired by an Oak tree that stood outside of Gosling’s office. Oak tree is a symbol of strength and chosen as a national tree of many countries like U.S.A, France, Germany etc.

History of Java

- But in 1995, team had decided to change the name of language, because Oak name was already a trademark by Oak technologies.
  - New Name was Java

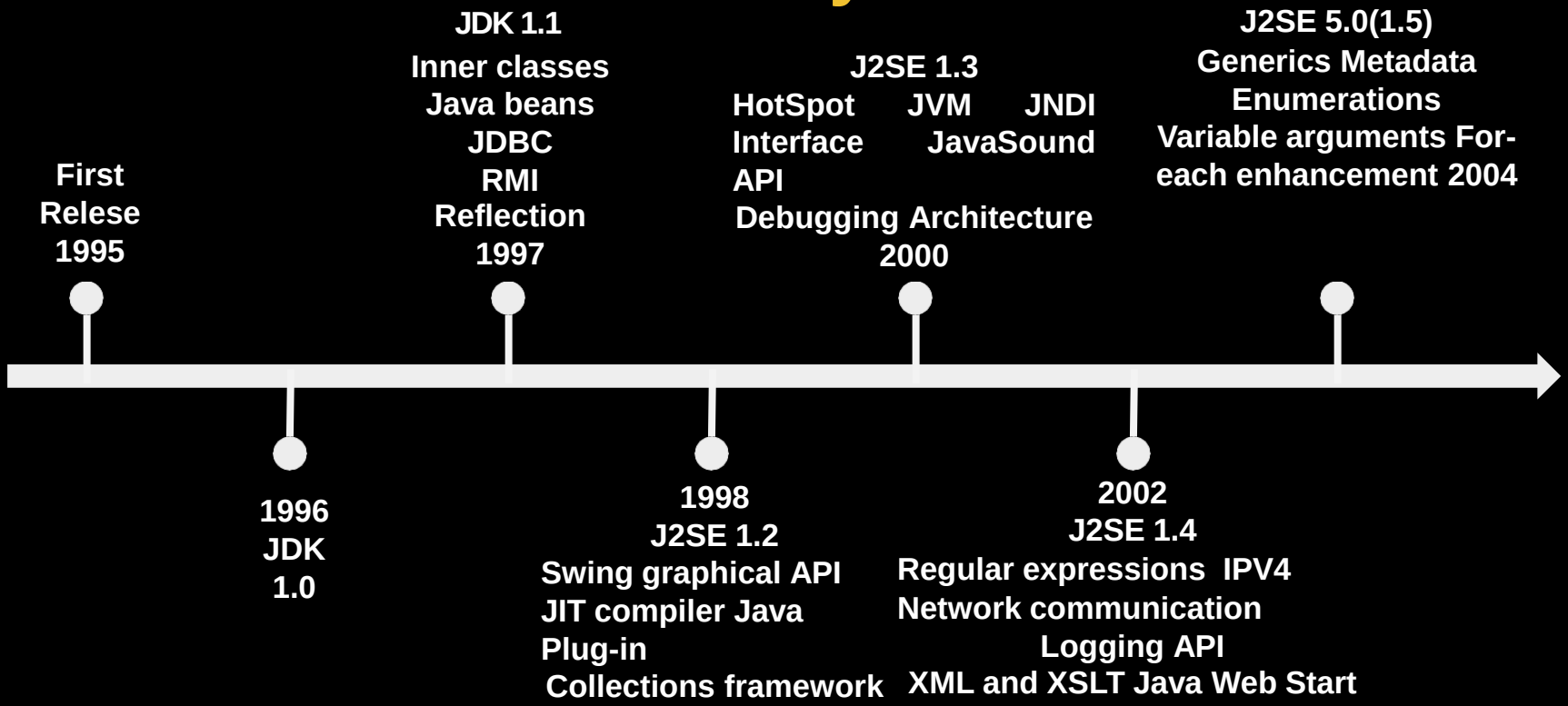
  The suggested words were “dynamic”, “revolutionary”, “silk”, “joit”, “DNA” etc

  According to James Gosling “Java and Silk” were two top choices. Since java was so unique, most of the team members preferred java

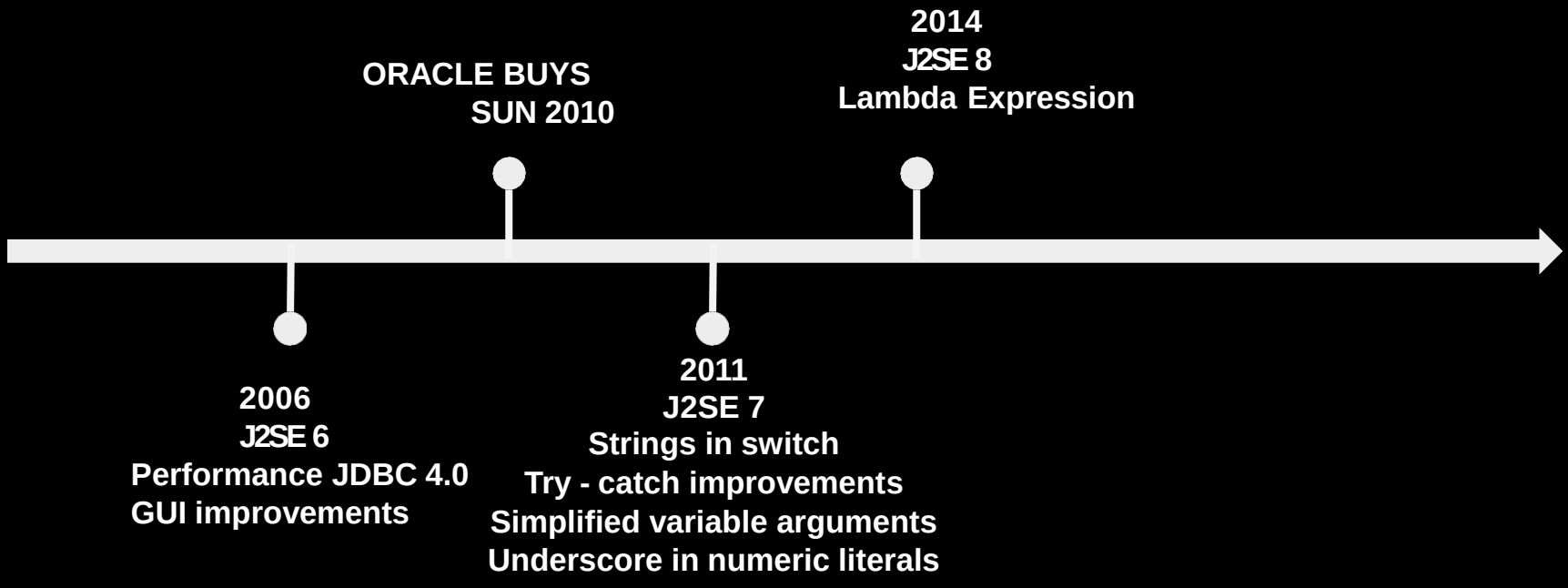
History of Java

- **Java**: Java is an island of Indonesia where first coffee was produced (called java coffee)
- Java coffee consumed in large quantities by language creators.
- Java is just a name not an acronym.

History of Java



History of Java



Features of Java or Java Buzzwords

- Simple
- Object - Oriented
- Platform independent
- Secured
- Robust
- Architecture neutral
- Portable
- Dynamic

Features of Java

- Interpreted
- High Performance
- Multi Threaded
- Distributed

Features of Java

- **Simple**: Java is considered as one of simple language because it doesn’t have complex features like operator overloading, Multiple inheritance, pointers and explicit memory management.

Features of Java

- **Object Oriented :**

Basic concepts of Object Oriented programming are

1. **Object**
2. **Class**
3. **Inheritance**
4. **Polymorphism**
5. **Abstraction**
6. **Encapsulation**

Features of Java

- **Platform Independent :**

A platform is the hardware or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides software-based platform.

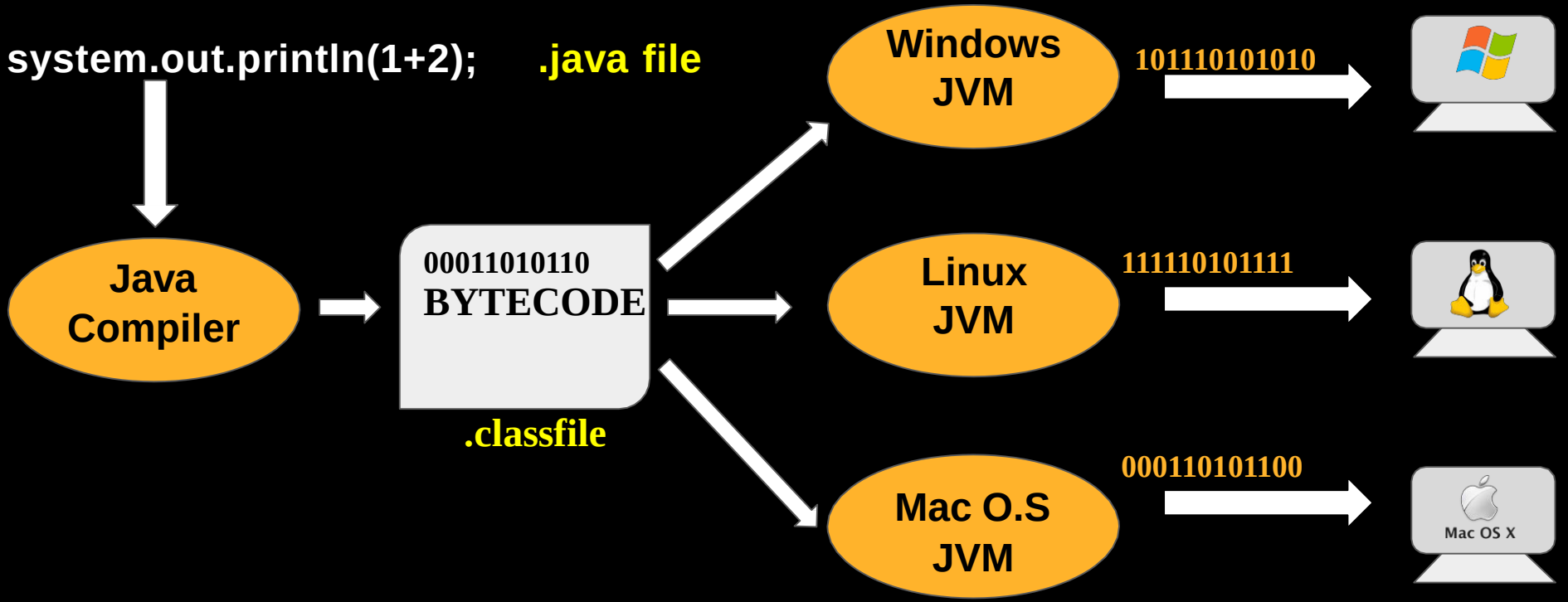
Features of Java

Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc.

Java code is compiled by the compiler and converted into bytecode.

This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).

Features of Java



Features of Java

- **Robust**: Two main problems that cause program failures are memory management mistakes and mishandled runtime errors. Java handles both of them efficiently.
- Memory management mistakes can be overcome by garbage collection. Garbage collection is automatic deallocation of objects which are no longer needed. Mishandled runtime errors are resolved by Exception Handling procedures.

Features of Java

Portable:

➢When we compile the java program, the compiler of java generates .class file that contains the Bytecodes of your java program.

➢The generated Bytecodes are secure and can run on any machine (portable) which has JVM.

Features of Java

➢ **Architecture - In neutral:**

java there is no implementation dependent feature e.g. size of primitive types is fixed.

In C programming, float data type occupies 4 bytes of memory for 32-bit architecture and 8 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

Features of Java

- **Multi threaded**: Java supports multithreading. It enables a program to perform several tasks simultaneously.

- **Interpreted and High Performance**: Java interpreter converts the bytecode into processor readable binary code. For very high performance it uses a Just-in-time compiler.

Features of Java

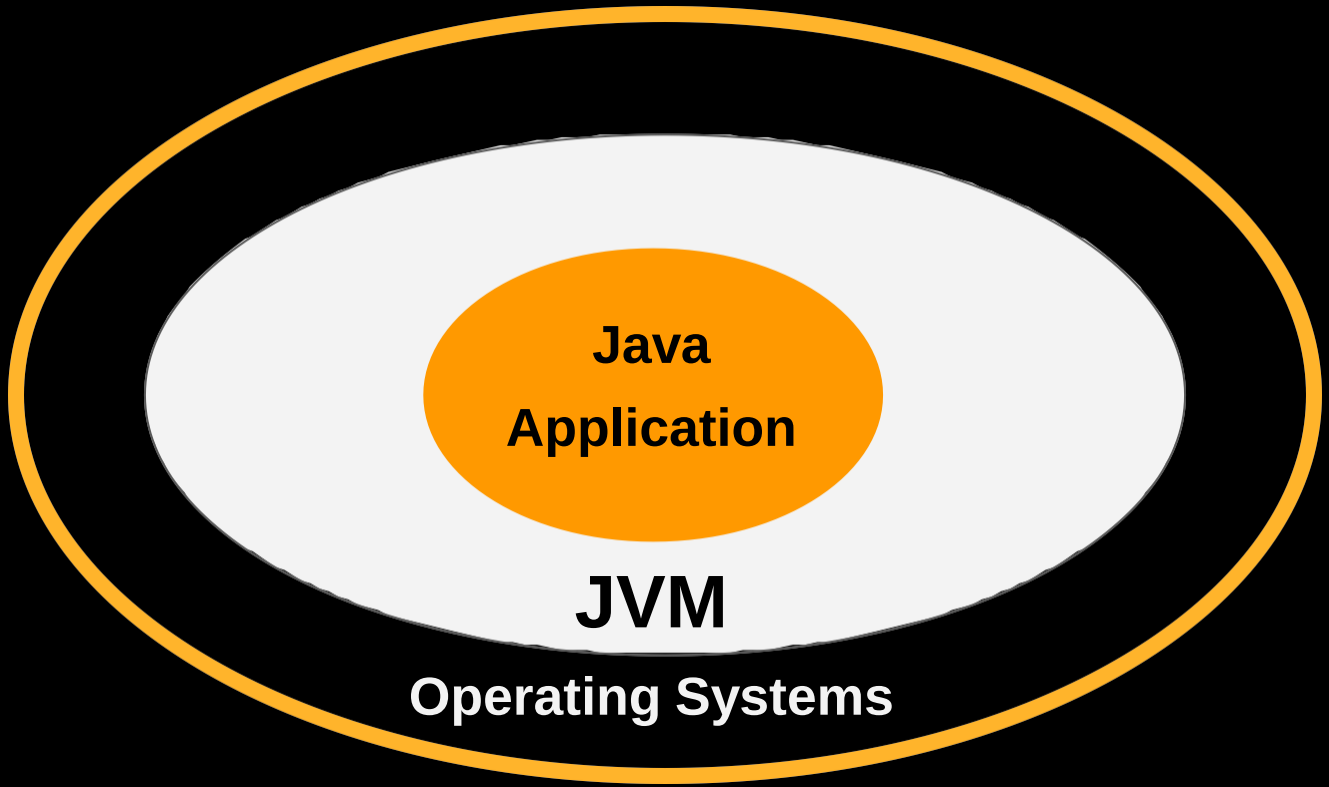
- **Distributed: RMI(Remote Method Invocation) and EJB(Enterprise Javabeans) are used for creating distributed applications in java.**
- **In simple words: The java programs can be distributed on more than one systems that are connected to each other using internet connection. Objects on one JVM (java virtual machine) can execute procedures on a remote JVM.**

Difference Between JDK, JVM and JRE

Java Virtual Machine (JVM)

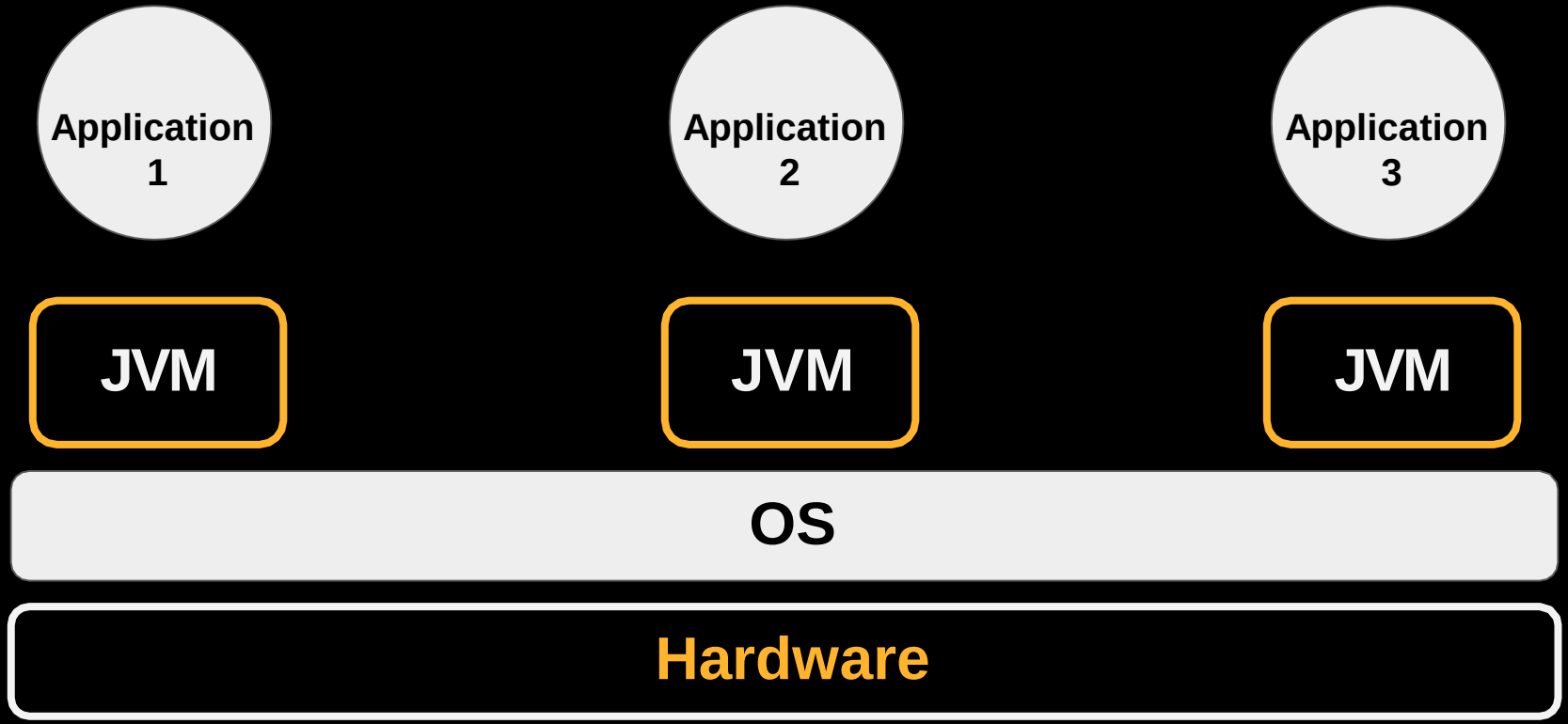
- JVM is the heart of Java programming Language.
- As we all aware when we compile a java file, output is not an “.exe” but it’s a ‘.class’ file. ‘.class’ file consists of Java bytecode.
- Java Virtual Machine interprets the bytecode into the machine code depending upon the underlying operating system and Hardware combination.

Java Virtual Machine (JVM)



Lifetime of a Java Virtual Machine

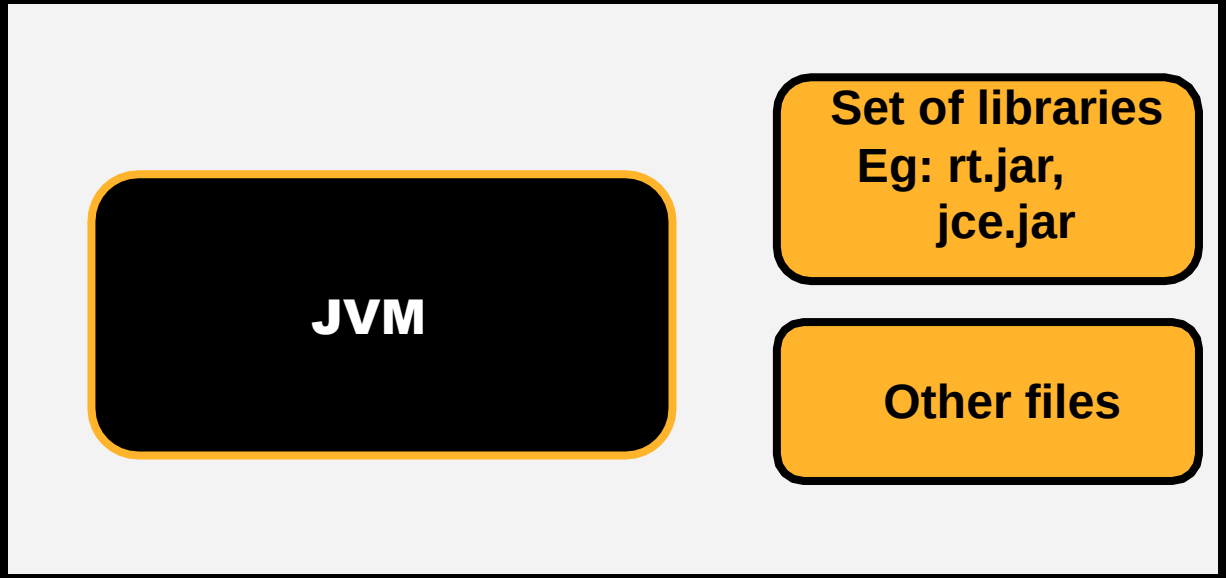
- When a java application starts, a runtime instance of JVM is created. When the application completes, the instance terminates.
- If you start three java applications at the same time, on the same computer, you will get three java virtual machine instances running
- Each java application runs inside its own java virtual Machine.


Application 1
Application 2
Application 3
JVM
JVM
JVM
OS
Hardware

Java Runtime Environment

- Java runtime environment contains JVM, class libraries and other supporting files.
- It doesn’t contain any development tools such as compiler, debugger etc.
- Actually JVM runs the program, and it uses the class libraries, and other supporting files provided in Java runtime environment.

Java Runtime Environment



JAVA RUNTIME ENVIRONMENT

JAVA DEVELOPMENT KIT

- Java development kit provides all necessary tools to develop any java application as well as runs it.

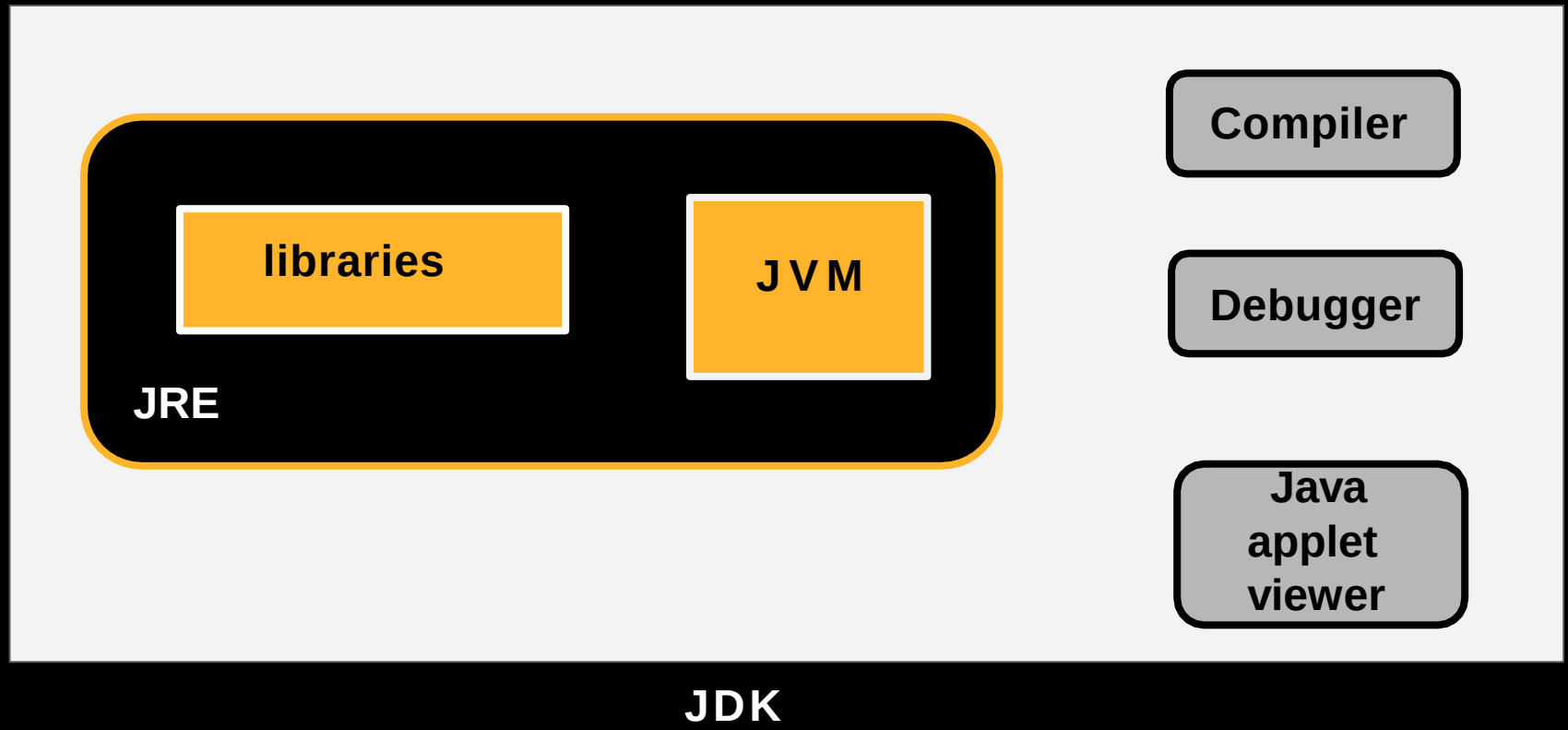
- JDK = Tools required to develop java application
  +
  Java Runtime Environment

JAVA DEVELOPMENT KIT

- The tools provided by JDK to develop java application are listed below

- Java Compiler
- Java Debugger
- Java appletviewer etc.

Java Development Kit



Creating Java Program

```
class FirstProgram {
public static void main(String args[]) {
System.out.println("First Java Program");
}
}
```

Compiling a Java Program

- A compiler is an application that translates programs from the Java language to a language more suitable for executing on the computer.

- It takes a text file with the .java extension as input (your program) and produces a file with a .class extension (the computer-language version).

Compiling a Java Program

- To compile FirstProgram.java type the following command at the terminal.

```
javac FirstProgram.java
```

Executing a java program

- Once you compile your program, you will get .class file. Now, we can execute the .class file by following command

```
java FirstProgram
```

OUTPUT: First Java Program

Understanding First Java Program

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents startup of the program.

Understanding First Java Program

- **static** is a keyword, if we declare any method as static, it is known as static method.
  - The core advantage of static method is that there is no need to create object to invoke the static method.
  - The main method is executed by the JVM, so it doesn't require to create object to invoke the main method. So it saves memory.

Understanding First Java Program

- **String[ ] args** is used for command line argument.

- **System.out.println()** is used print statement. We will learn about the internal working of System.out.println statement later.

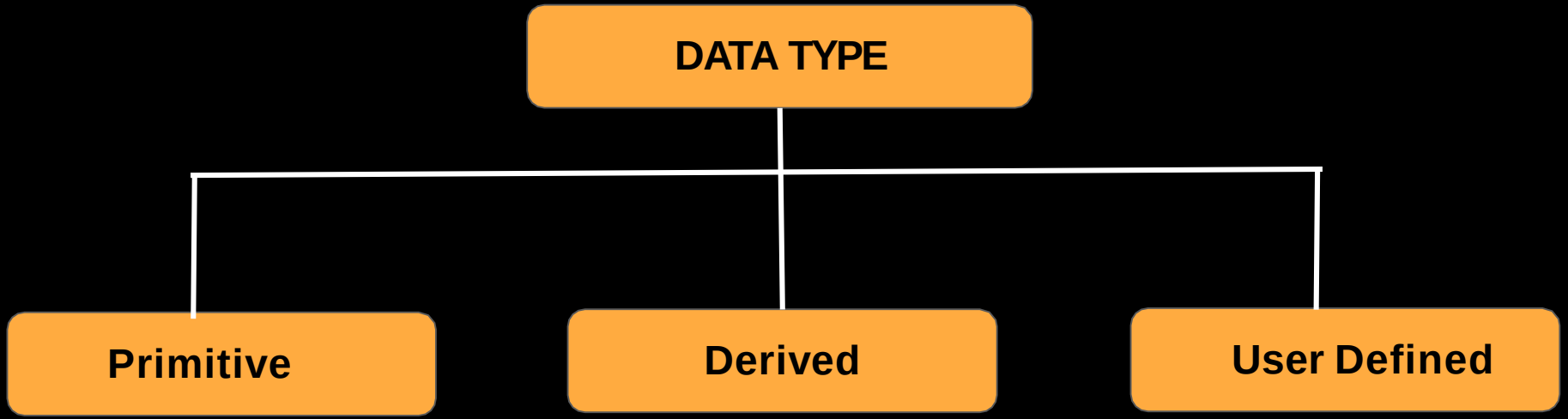
DATA TYPES IN JAVA

Data Types

- Data type is a special keyword used to allocate sufficient memory space for the data.
- In general every programming language is containing three categories of data types. They are

1. Fundamental or primitive data types
2. Derived data types
3. User defined data types.

Data Types



Primitive Data Types

- Primitive data types are those whose variables allows us to store only one value but they never allows us to store multiple values of same type.

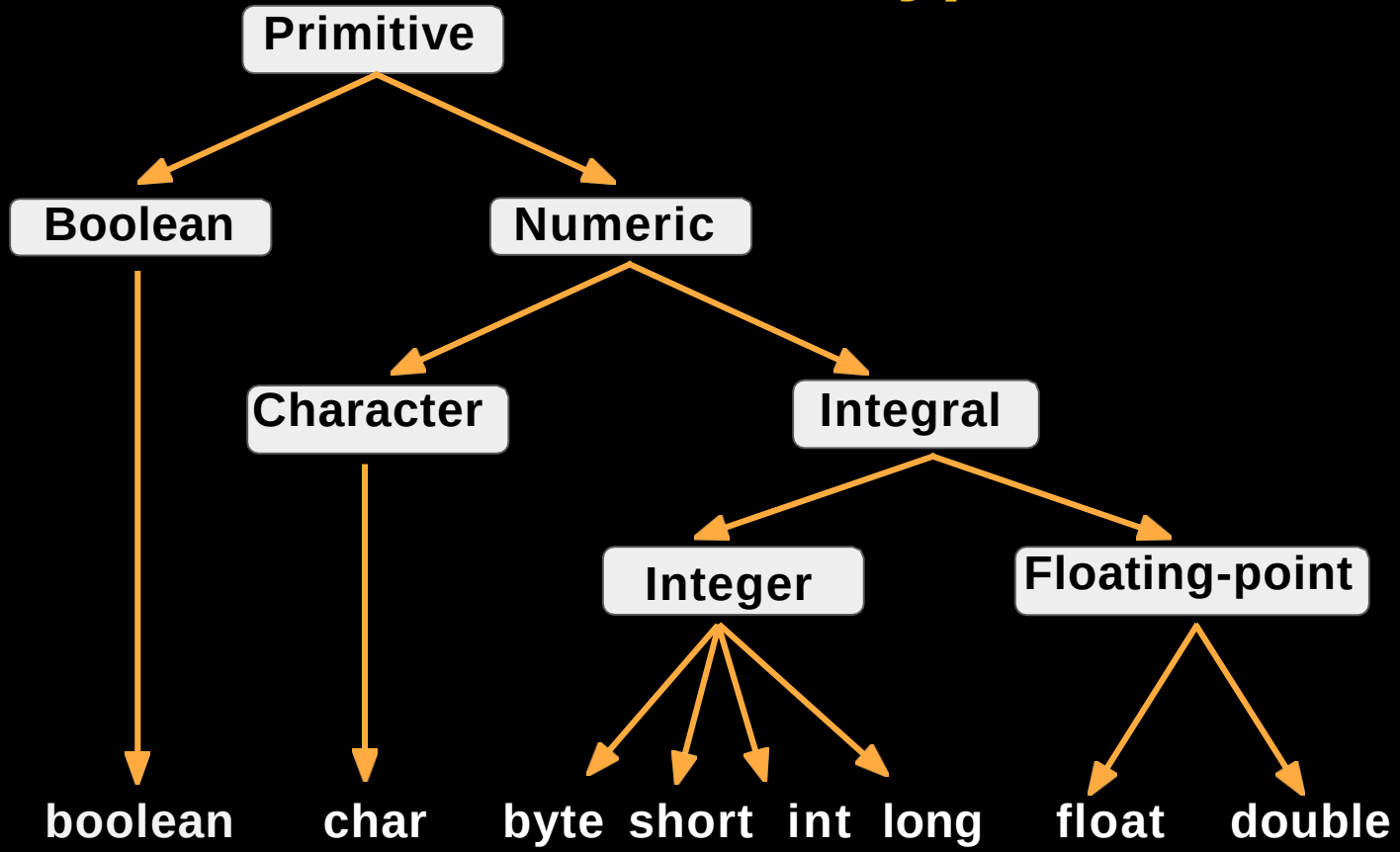
Example:

```
int a;
a = 10; // valid
a = 10, 20, 30; // not valid
```

Primitive Data Types

- There are eight built-in types supported by Java to support Four Integer Type, Two Float Type, One character type, and boolean type. All primitive or basic data types hold numeric data that is directly understood by system.

Primitive Data Types



Integer Type

- The integer types in Java are byte, short, int, and long. These four types differ only in the number of bits and, therefore, in the range of numbers each type can represent.
- All integral types represent signed numbers. There is no unsigned keyword as there is in C and C++.

Type	Contains	Default	Size	Range
byte	Signed integer	0	8 bits	-128 to +127
short	Signed integer	0	16 bits	-32768 to +32767
int	Signed integer	0	32 bits	-2147483648 to 2147483647
long	Signed integer	0	64 bits	-9223372036854775808 to 9223372036854775807

Boolean Type

- The boolean type represents a truth value. There are only two possible values of this type, representing the two boolean states: on or off, yes or no, true or false.
- Java reserves the words **true** and **false** to represent these two boolean values.
- **Note**: boolean values can never be converted to or from other data types

Boolean Type

Type	Contains	Default	Size	Range
boolean	true or false	false	1 bit	NA

Float Type

- The float types in Java are float and double. These two types differ only in the number of bits and, therefore, in the range of numbers each type can represent.
- float is a 32-bit, single-precision floating-point value, and double is a 64-bit, double-precision floating-point value.

Float Type

Type	Contains	Default	Size	Range
float	IEEE 754 floating point	0.0	32 bits	±1.4E-45 to ±3.4028235E+38
double	IEEE 754 floating point	0.0	64 bits	±4.9E-324 to ±1.7976931348623157E+308

Char Type

- In Java, the data type used to store characters is **char**.
- Character is 16 bits wide in Java.
- Java uses **Unicode** to represent characters.
- Java support lot of Unicode symbols from many more human languages for this purpose, it requires 16 bits.
- The range of a char is 0 to 65,536.
- There are no negative chars.

Char Type

Type	Contains	Default	Size	Range
char	Unicode character	\u0000	16 bits	\u0000 to \uFFFF

Derived Data Types

- Derived data types are those whose variables allow us to store multiple values of same type. But they never allows to store multiple values of different types.
- In general derived data type can be achieve using array.
- Example:

```
int a[ ] = {10,20,30}; // valid
int b[ ] = {100, 'A', "ABC"}; // invalid
```

User Defined Data types

- User defined data types are those which are developed by programmers by making use of appropriate features of the language.
- User defined data types related variables allows us to store multiple values either of same type or different type or both.
- This is a data type whose variable can hold more than one value of dissimilar type, in java it is achieved using class concept.

User Defined Data types

- **Note: In java both derived and user defined data types combined name as reference data type.**
- **In C language, user defined data types can be developed by using struct, union, enum etc.**
- **In java programming user defined data type can be developed by using the features of classes and interfaces.**

Type Casting

- The process of converting one data type to another is called casting.
- Casting is often necessary when a function returns a data in different form than what we need.
- Under certain circumstances Type conversion can be carried out automatically, in other cases it must be "forced" manually (explicitly)

Type Casting

- The storage size of the types while casting is very important otherwise the data loss will occur.
- For example, suppose we cast a long to an int. A long is a 64-bit value and an int is a 32-bit value. When casting a long to an int, the compiler truncates the upper 32 bits of the long value so as to fit into the 32-bit int.
- If the upper 32 bits of the long contain any useful information, then data loss will occur.

Type Casting

- In some cases, the data type of the expression is changed automatically to the variable's data type. For example, suppose that i is an integer variable declared as follows:

```
int i = 10 ;
```

- Even though d is a variable of type double, the following assignment is valid:

```
double d = i *10; // valid,
(i*10) is converted to type  double
```

Type Casting

- Java changes (i*10) from int to double, and then assigns it to d.
- This conversion does not cause any loss of information because the double data type is 64 bits wide and thus is wider than the int data type that has 32 bits. This is called a widening conversion because a narrower type is converted to a wider type, and it takes place implicitly.

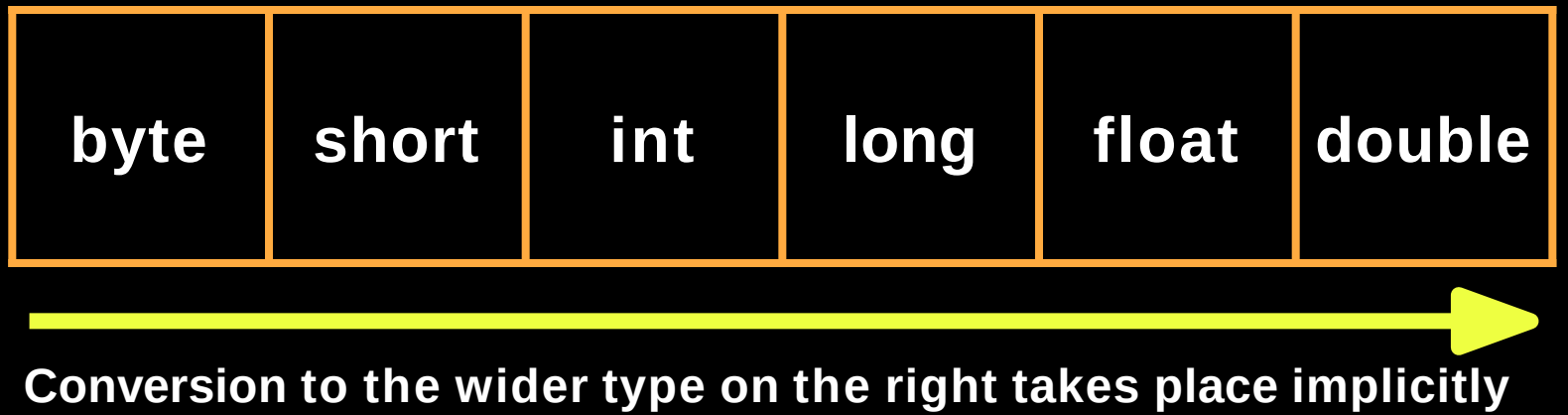
Type Casting

- some more examples of when widening conversions occur:
  - short s = 10;
  - int num= s;

  - float f = 10.5f;
  - double d = f;

Type Casting

- Following Figure summarizes the conversion rules: A conversion occurs from a given data type to any of the data types on its right.



Type Casting

- Assigning a value of type double to an int causes a compilation error:

```
double d = 100.5;
int i = d; // error!
```

- Java does not allow this assignment because a double can hold a larger range of values than an int, and it could result in a loss of data.

Type Casting

- To allow this type of assignment, it is necessary for the programmer to force a conversion from double to int using a cast.
- For example, a cast is used here to force a conversion of the variable d to type int and then assign it to i:

```
int i = (int) d; // i = 100
```

- Note: The above assignment truncates (not rounds) the fractional part so that it assigns 100 to i.

Overview of Type Casting

- **Automatic Conversion**: In Java type conversions are performed automatically when the type of the expression on the right hand side of an assignment operation can be safely promoted to the type of the variable on the left hand side of the assignment. Thus we can safely assign:

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

Type Casting

- **Explicit Conversion (Casting):** we cannot automatically convert a long to an int because the first requires more storage than the second and consequently information may be lost.
- To force such a conversion we must carry out an explicit conversion (assuming that the long integer will fit into a standard integer). This is done using a process known as a **type cast**

Type Casting

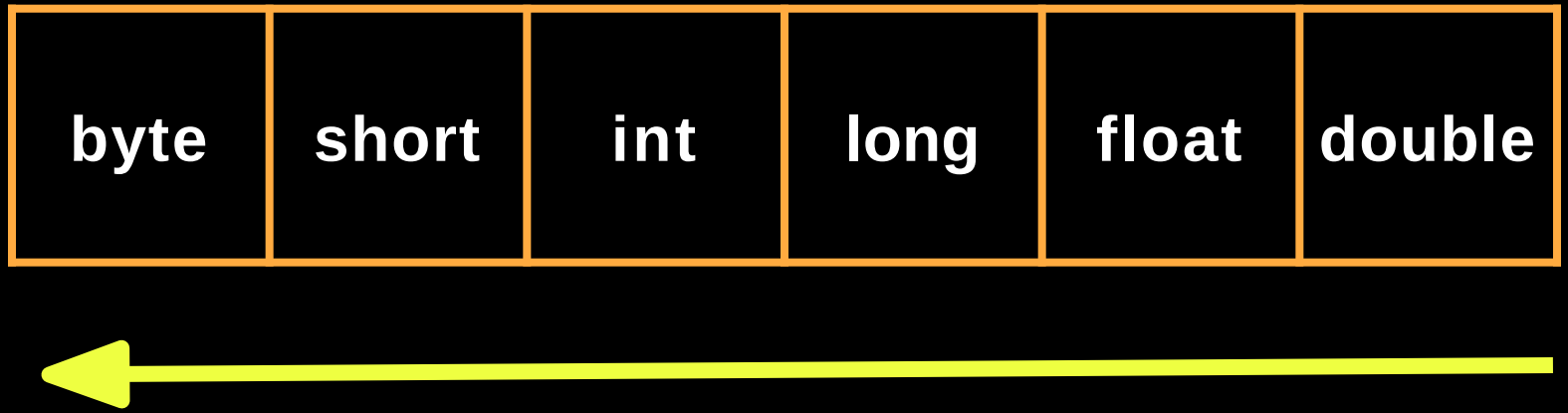
myInteger = (int) myLongInteger

This tells the compiler that the type of myLongInteger must be temporarily changed to an int when the given assignment statement is processed. Thus, the cast only lasts for the duration of the assignment.

Type Casting

```
public class Main {
    public static void main(String[ ] args) {
        double myDouble = 9.78;
        int myInt = (int) myDouble; // Manual casting: double to int
        System.out.println(myDouble);   // Outputs 9.78
        System.out.println(myInt);      // Outputs 9
    }
}
```

Type Casting



Conversion to the narrower type on the right takes place explicitly

HOW TO GET INPUT FROM USER IN JAVA

Java Scanner Class

Java Scanner class allows the user to take input from the console. It belongs to java.util package. It is used to read the input of primitive types like int, double, long, short, float, and byte. It is the easiest way to read input in Java program.

Syntax

```
Scanner sc=new Scanner(System.in);
```

The above statement creates a constructor of the Scanner class having System.inM as an argument. It means it is going to read from the standard input stream of the program. The java.util package should be import while using Scanner class.

Methods of Java Scanner Class

Java Scanner class provides the following methods to read different primitives types:

Method	Description
int nextInt()	It is used to scan the next token of the input as an integer.
float nextFloat()	It is used to scan the next token of the input as a float.
double nextDouble()	It is used to scan the next token of the input as a double.
byte nextByte()	It is used to scan the next token of the input as a byte.
String nextLine()	Advances this scanner past the current line.
boolean nextBoolean()	It is used to scan the next token of the input into a boolean value.
long nextLong()	It is used to scan the next token of the input as a long.
short nextShort()	It is used to scan the next token of the input as a Short.
BigInteger nextBigInteger()	It is used to scan the next token of the input as a BigInteger.
BigDecimal nextBigDecimal()	It is used to scan the next token of the input as a BigDecimal.

Integer input from user

```
import java.util.*;
public class UserInputDemo
{
public static void main(String[ ] args)
{
Scanner sc= new Scanner(System.in);    //System.in is a standard input stream
System.out.print("Enter first number- ");
int a= sc.nextInt();
System.out.print("Enter second number- ");
int b= sc.nextInt();
System.out.print("Enter third number- ");
int c= sc.nextInt();
int d=a+b+c;
System.out.println("Total= " +d);
}
}
```

JAVA CONTROL STATEMENTS

Control Flow Statements

Decision Making statements

- **if statements**
- **switch statement**

Loop statements

- **do while loop**
- **while loop**
- **for loop**
- **for-each loop**

Jump statements

- **break statement**
- **continue statement**

If Statement

Use the if statement to specify a block of Java code to be executed if a condition is true.

Ex-1

```
if (20 > 18) {
System.out.println("20 is greater than 18");
}
```

Ex-2

```
int x = 20;
int y = 18;
if (x > y) {
System.out.println("x is greater than y");
}
```

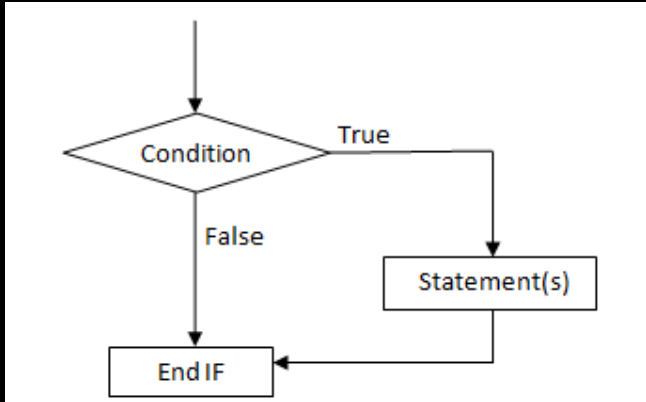
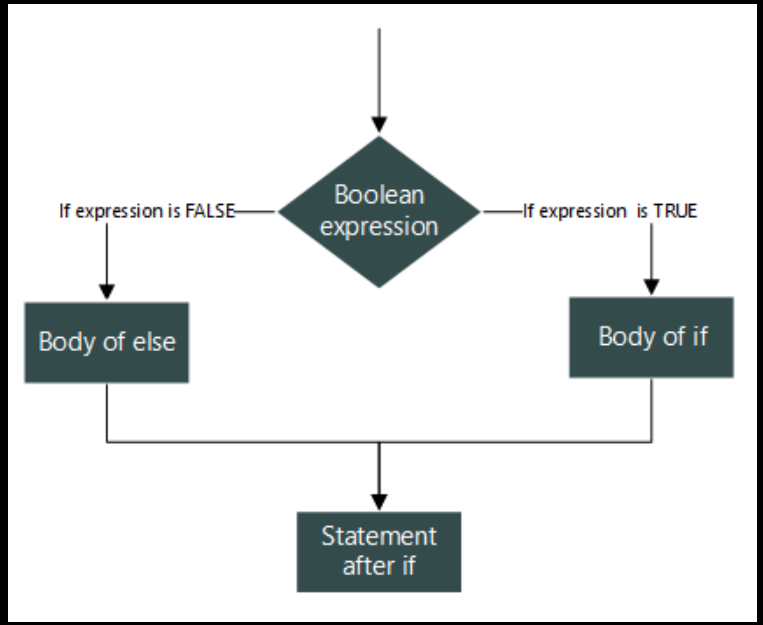


fig: Flowchart for if statement

The if-else Statement

Use the **else** statement to specify a block of code to be executed if the condition is **false**.

```
int time = 20;
if (time < 18) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
```



The else if Statement

Use the **else if** statement to specify a new condition if the first condition is **false**.

```
int time = 22;
if (time < 10) {
  System.out.println("Good morning.");
} else if (time < 20) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
```

The else if Statement

Use the **else if** statement to specify a new condition if the first condition is **false**.

```
int time = 22;
if (time < 10) {
  System.out.println("Good morning.");
} else if (time < 20) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
```

Output:-Good evening.

Java Switch Statements

Use the `switch` statement to select one of many code blocks to be executed.

```
int day = 1;
switch (day) {
  case 1:
    System.out.println("Today is Saturday");
    break;
  case 2:
    System.out.println("Today is Sunday");
    break;
  default:
    System.out.println("Looking forward to the Weekend");
}
```

Java Switch Statements

Use the `switch` statement to select one of many code blocks to be executed.

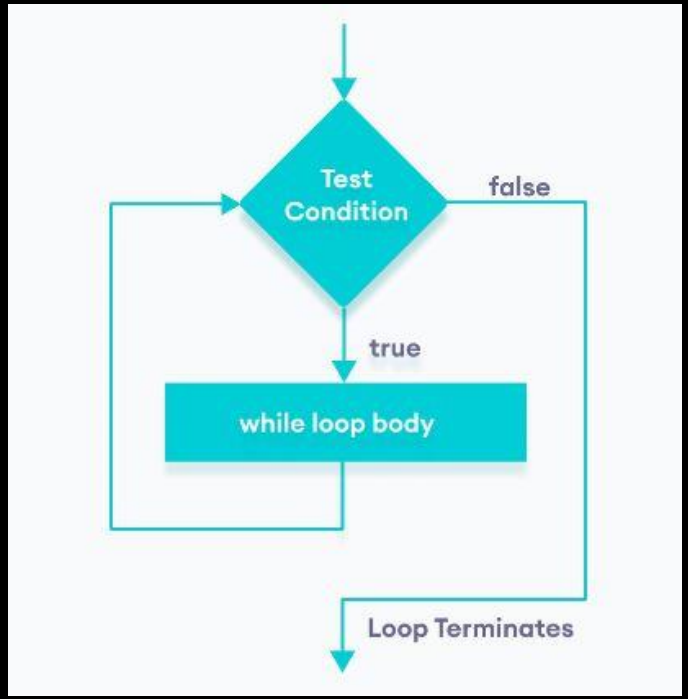
```
int day = 1;
switch (day) {
  case 1:
    System.out.println("Today is Saturday");
    break;
  case 2:
    System.out.println("Today is Sunday");
    break;
  default:
    System.out.println("Looking forward to the Weekend");
}
```

Output:-Today is Saturday

The While Loop

The `while` loop loops through a block of code as long as a specified condition is true.

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
```

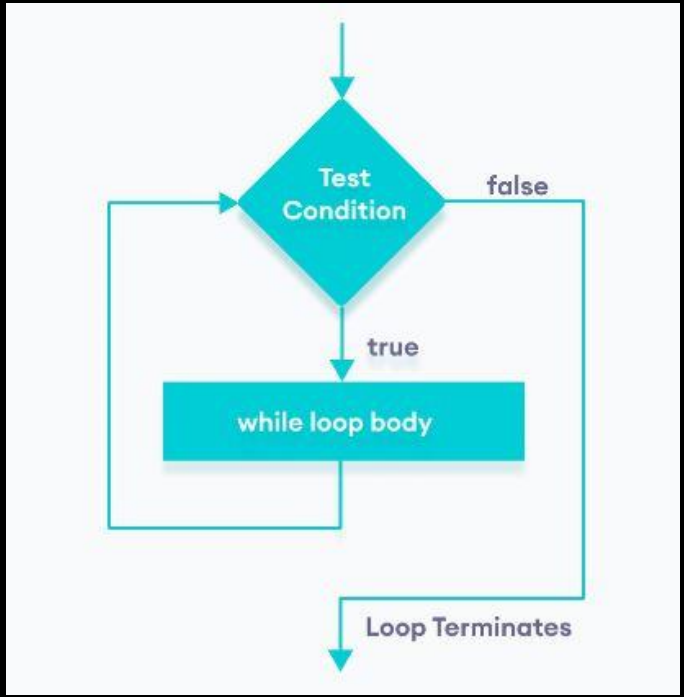


The While Loop

The `while` loop loops through a block of code as long as a specified condition is true.

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
```

Output:- 0 1 2 3 4

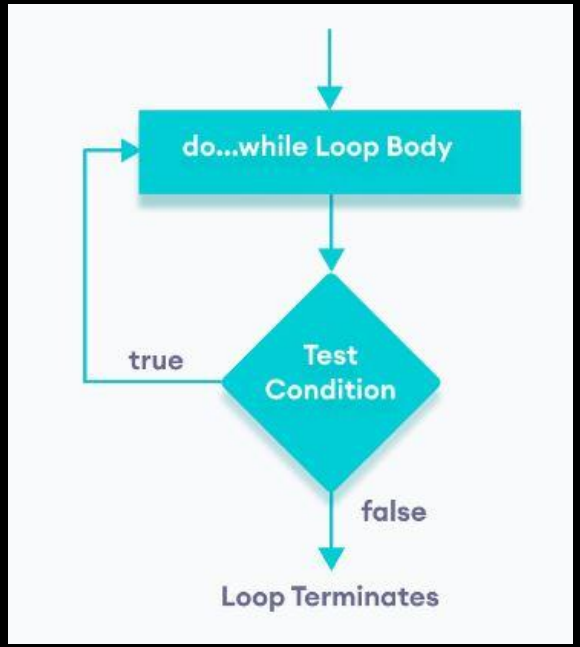


The Do/While Loop

The `do/while` loop is a variant of the `while` loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

```
int i = 0;
do {
  System.out.println(i);
  i++;
}
while (i < 5);
```

Output:- 0 1 2 3 4

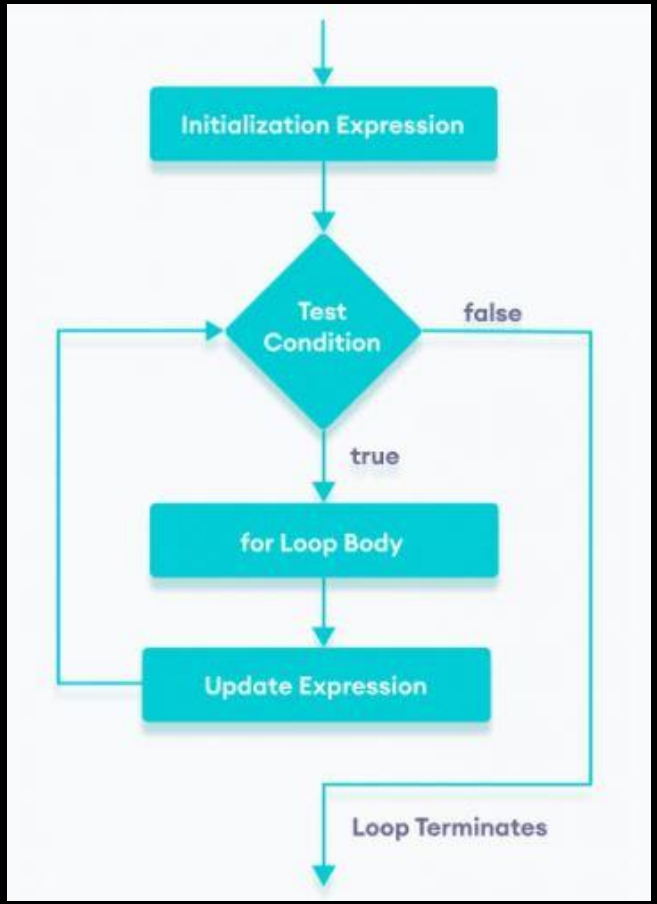


Java For Loop

```
for (int i = 0; i < 5; i++)
{
  System.out.println(i);
}
```

```
for (int i = 0; i <= 10; i = i + 2)
{
  System.out.println(i);
}
```

Output:- 0 2 4 6 8 10



ARRAY

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type with square brackets:

```
int[ ] myNum;
```

```
int[ ] myNum = {10, 20, 30, 40};

String[ ]cars = {"Volvo", "BMW", "Ford", "Mazda"};
```

Access the Elements of an Array

```
String[ ] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
```

Output:- Volvo

Change an Array Element

```
String[ ] cars = {"Volvo", "BMW", "Ford", "Mazda"};
cars[0] = “Audi";
System.out.println(cars[0]);
```

Output:- Audi

Array Length

To find out how many elements an array has, use the length property:

```
String[ ] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars.length);
```

Output:- 4

Java Arrays Loop

```
String[ ] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.length; i++) {
  System.out.println(cars[i]);
}
```

Output:- Volvo, BMW, Ford, Mazda

For each loop

```
String[ ] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars)
{
  System.out.println(i);
}
```

Output:- Volvo, BMW, Ford, Mazda

Java Multi-Dimensional Arrays

A multidimensional array is an array of arrays.

To create a two-dimensional array, add each array within its own set of curly braces:

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

Java Multi-Dimensional Arrays

```
int[ ][ ] myNumbers = { {1, 2, 3, 4}, {5, 6, 7,8} };
int x = myNumbers[1][2];
System.out.println(x);
```

Output:- 7

Java Multi-Dimensional Arrays

```
public class Main {
  public static void main(String[ ] args) {
        int[ ][ ] myNumbers = { {1, 2, 3, 4}, {5, 6, 7,8} };
            for (int i = 0; i < myNumbers.length; ++i) {
                        for(int j = 0; j < myNumbers[i].length; ++j) {
                                    System.out.println(myNumbers[i][j]);
                        }
              }
       }
}
```

OPERATORS

Operators

1. Arithmetic Operators.
2. The Bitwise Operators
3. Relational Operators
4. Boolean Logical Operators
5. Assignment Operators
6. The ? Operator
7. Operator Precedence

Arithmetic Operators

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Arithmetic Operators

```
public class IncrementDecrementQuiz
{
        public static void main(String[ ] args)
        {
                int i = 11;

                i = i++ + ++i;

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

Output:- 24

Arithmetic Operators

```
public class IncrementDecrementQuiz
{
        public static void main(String[ ] args)
        {
                int i = 11;

                int j = --(i++);

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

Output:- Error

Relational Operators

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Boolean Logical Operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10) !(x < 5 || x < 4) !(x < 5)

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
^=	x ^= 3	x = x ^ 3

The ? Operator

Java ternary operator is the only conditional operator that takes three operands. It's a one-liner replacement for the if-then-else statement and is used a lot in Java programming

Syntax:

Variable = Expression1 ? Expression2: Expression3

```
if(Expression1)
{
    variable = Expression2;
}
else
{
    variable = Expression3;
}
```

The ? Operator

```
public class IncrementDecrementQuiz
{
        public static void main(String[ ] args)
        {
                int p=0;

                p = 5<2 ? 4 : 3;

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

Output:- 3

The ? Operator

```
public class LargestNumberExample
{
        public static void main(String args[ ])
        {
                int x=69;
                int y=89;
                int z=79;
int largestNumber = (x > y) ? (x > z ? x : z) : (y > z ? y : z);
System.out.println("The largest numbers is:  "+largestNumber);
        }
}
```

Output:- 89