VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

CSE1007
Java Programming

Slot: B1+TB1
Module -I

Venue: AB1-810
Java Basics

Prof. Tulasi Prasad Sariki
SCSE, VIT, Chennai
www.learnersdesk.weebly.com

Course Contents:

- Java Technology
  - Programming Language
  - Platform
- Features of Java Language
- Java source file structure
- Basic programming constructs
- Arrays
  - One dimensional and Multidimensional
- Enhanced for loop
- String, StringBuffer & StringBuilder
- Wrapper classes

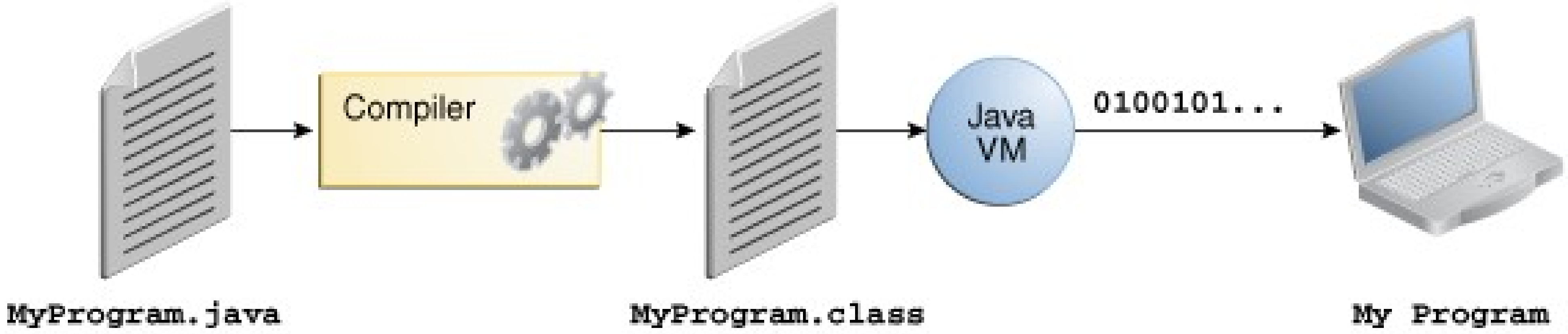
What is Java?

- Java technology is both a *programming language* and a *platform*.
- Java is a high-level programming language that can be characterized by its special features.
- Java is an object-oriented, cross platform, multipurpose programming language produced by Sun Microsystems, later acquired by Oracle Corporation.

Java Programming Language

- In the Java , all the source code is first written in plain text files ending with the *.java* extension.
- Those source files are then compiled into *.class* files by the *javac* compiler.
- A *.class* file does not contain code that is native to your processor; it instead contains bytecodes— the machine language of the Java Virtual Machine(Java VM).
- The *java* interpreter tool then runs your application with an instance of the Java Virtual Machine.

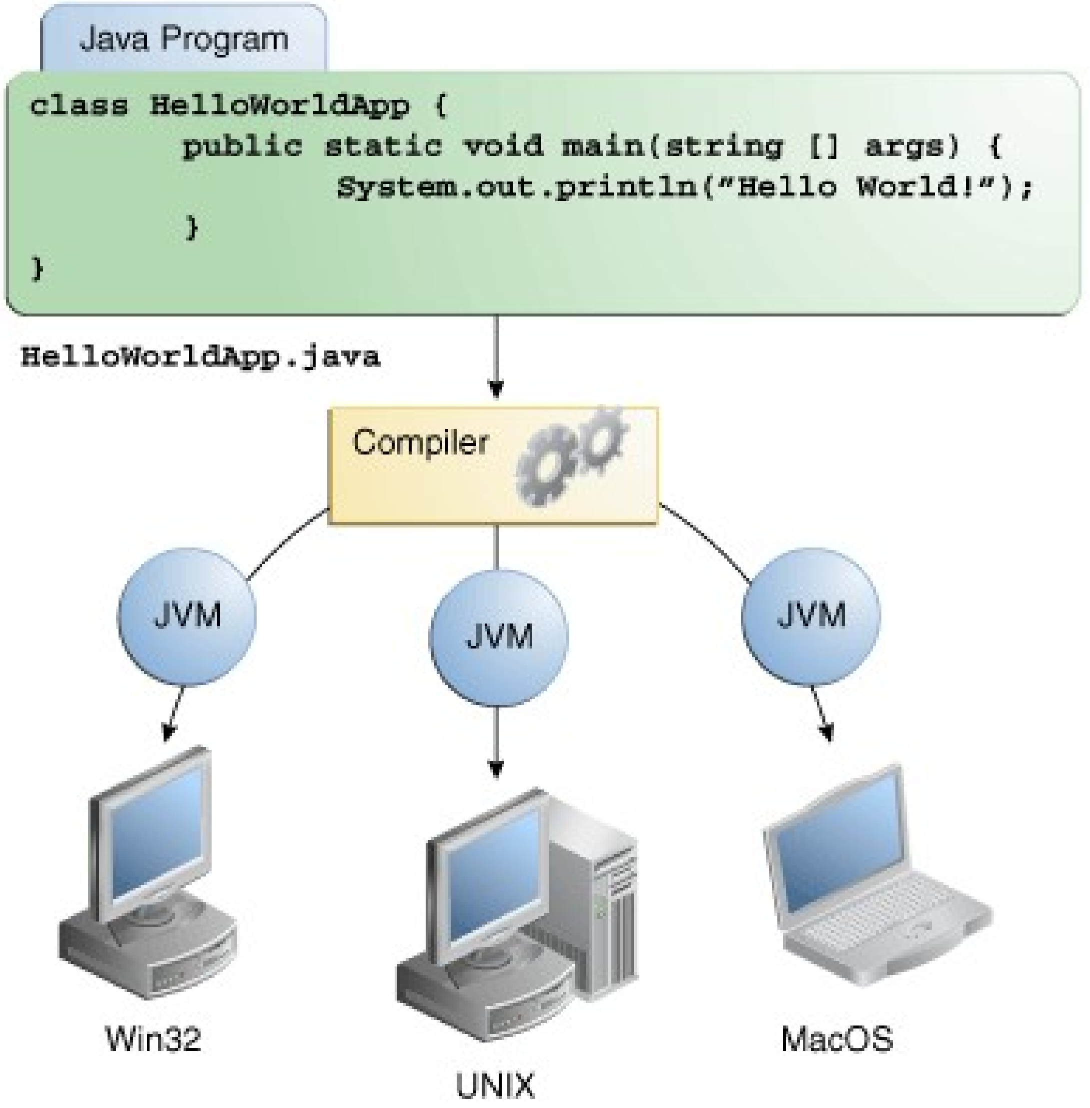
Overview of the S/W development process



Overview of the S/W development process

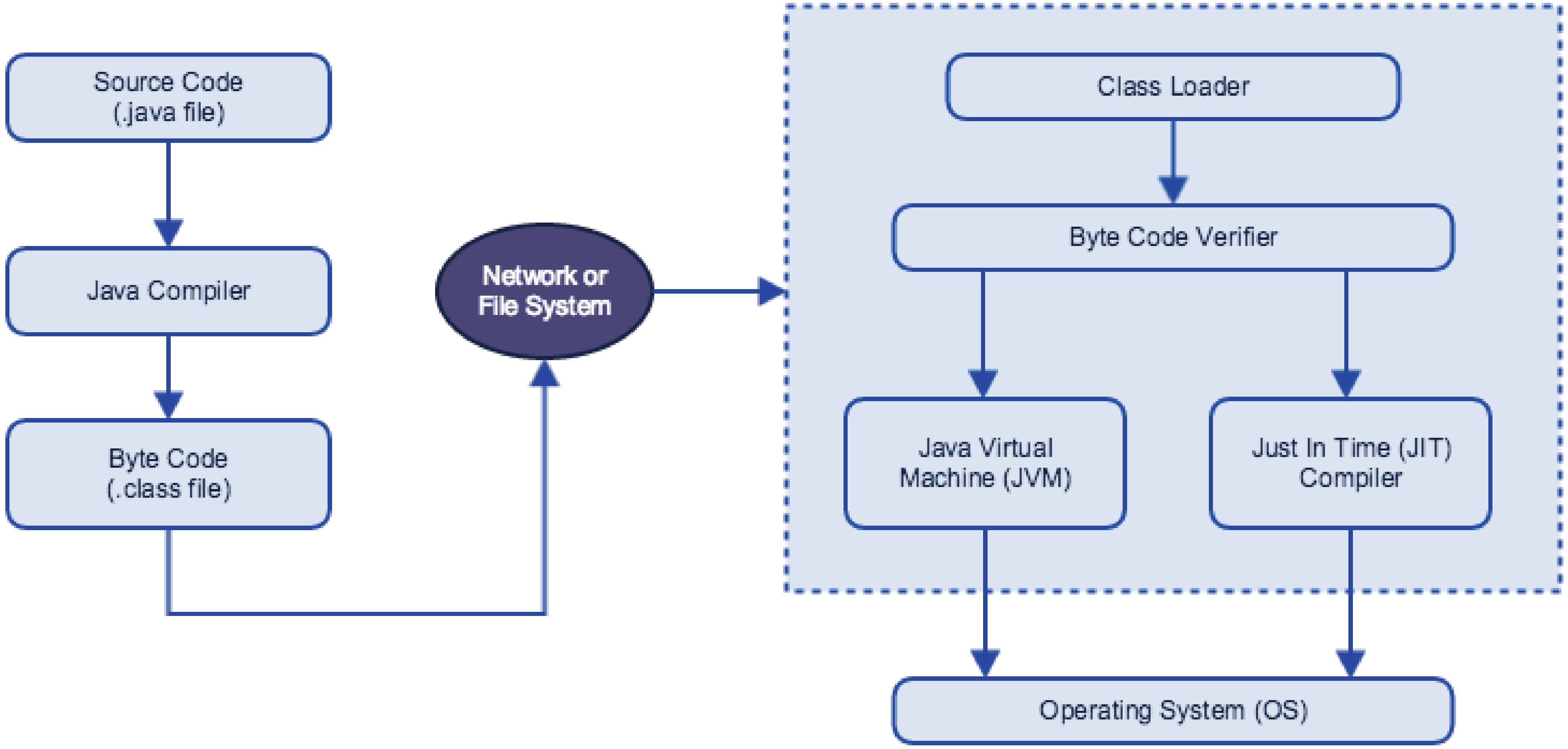
- Java VM is available on many different operating systems, the same .class files are capable of running on Microsoft Windows, the Solaris™ Operating System (Solaris OS), Linux, or Mac OS.
- Some virtual machines, such as the Java SE HotSpot , perform additional steps at run-time to give your application a performance boost. This includes various tasks such as finding performance bottlenecks and recompiling (to native code) frequently used sections of code.

Overview of the S/W development process



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

Overview of the S/W development process



The JIT compiler translates the Java bytecode into native processor instructions at run-time and caches the native code in memory during execution.

The Java Platform

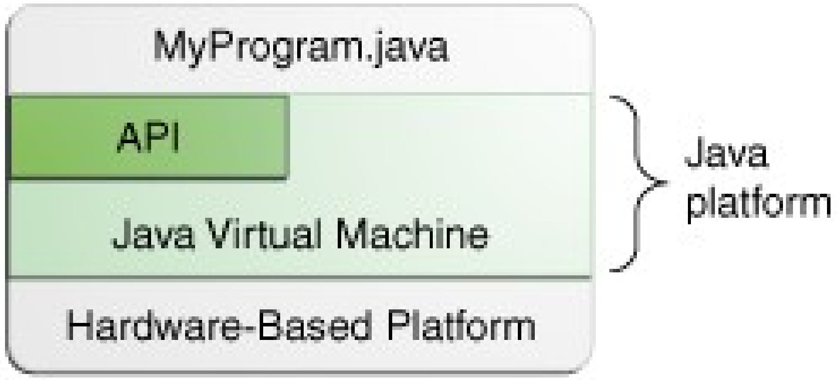
- A platform is the hardware or software environment in which a program runs (Microsoft Windows, Linux, Solaris OS, and Mac OS).
- Most platforms can be described as a combination of the operating system and underlying hardware.
- The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other hardware-based platforms.

The Java Platform

- The Java platform has two components:
  - The Java Virtual Machine
  - The Java Application Programming Interface (API)
- JVM is the base for the Java platform and is ported onto various hardware-based platforms.
- The API is a large collection of ready-made software components that provide many useful capabilities. It is grouped into libraries of related classes and interfaces; these libraries are known as packages.

The Java Platform

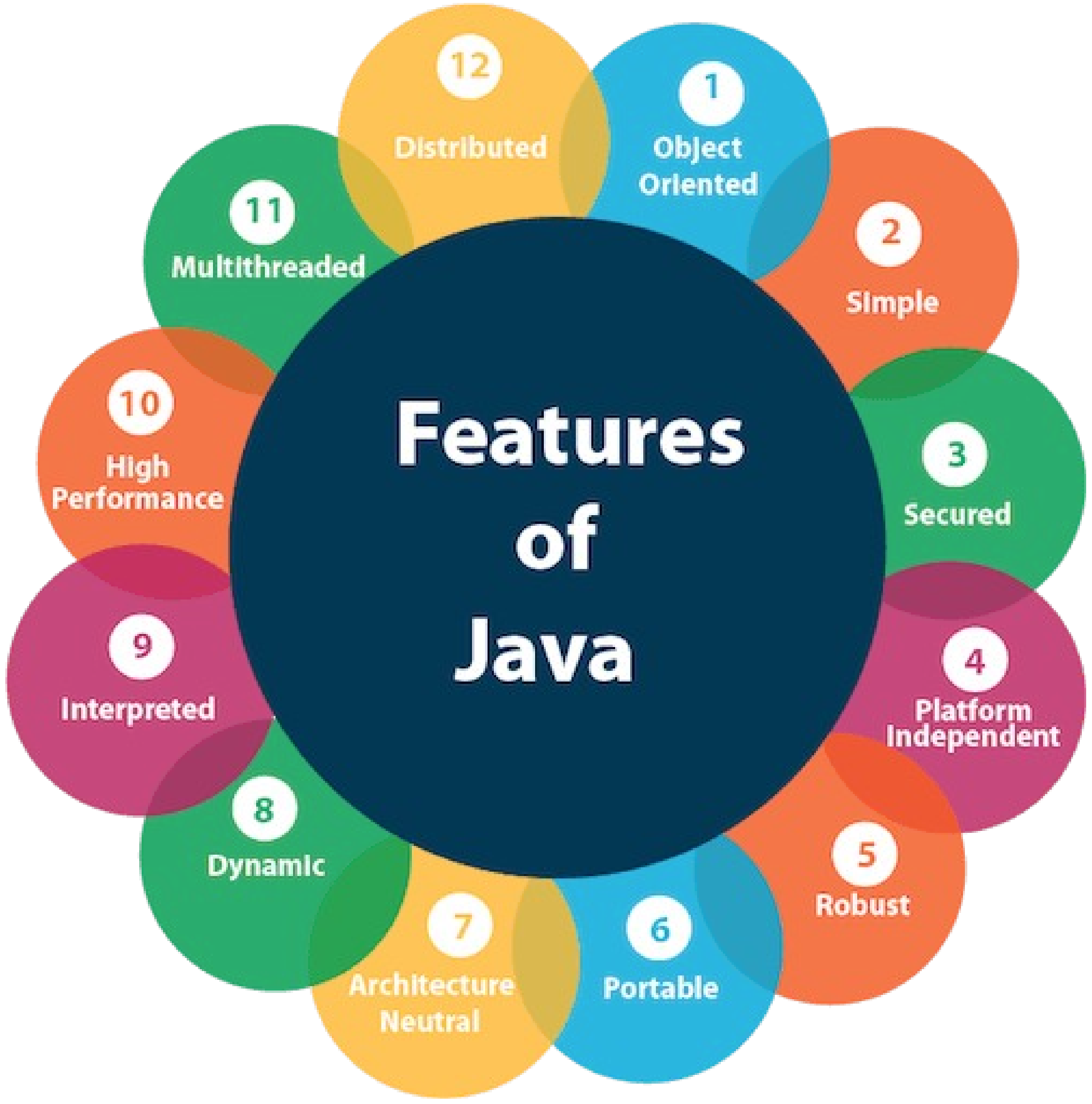
- The API and Java Virtual Machine insulate the program from the underlying hardware.



- As a platform-independent environment, the Java platform can be a bit slower than native code.
- However, advances in compiler and virtual machine technologies are bringing performance close to that of native code without threatening portability.

Features of Java

https://web.cs.wpi.edu/~kal/elecdoc/java/features.html



Features of Java

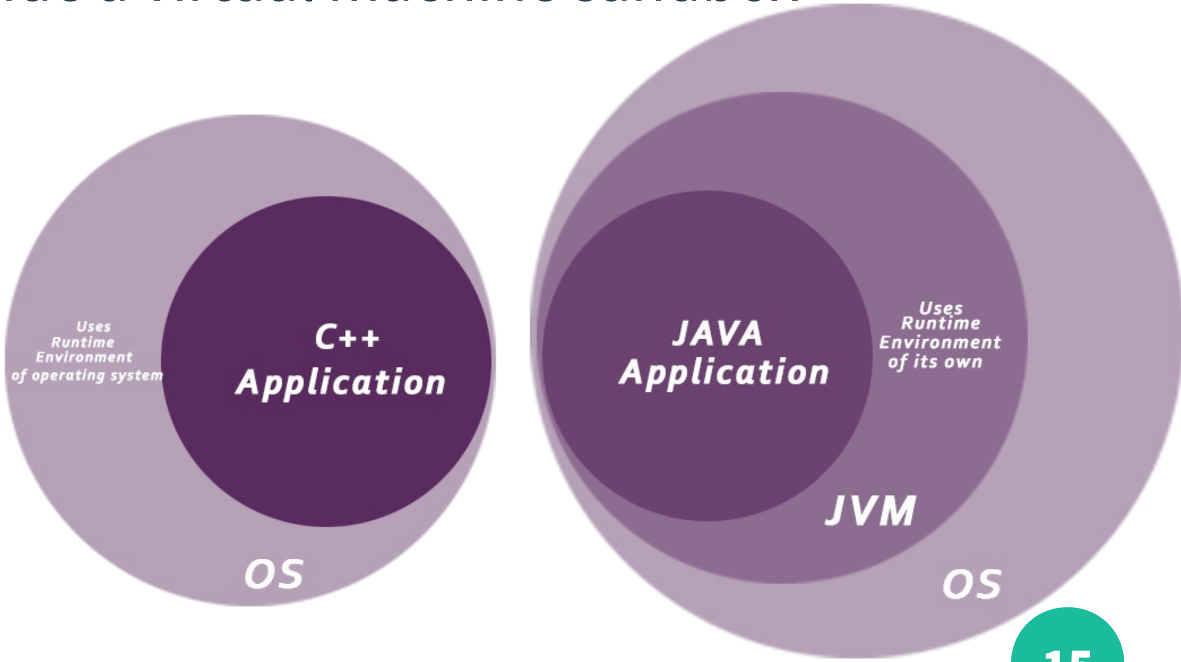
- Java is an object-oriented programming language. Everything in Java is an object. Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behavior.
- Object-oriented programming (OOPs) is a methodology that simplifies software development and maintenance by providing some rules.
- Basic concepts of OOPs are:
  - Object
  - Class
  - Inheritance
  - Polymorphism
  - Abstraction
  - Encapsulation

Features of Java

- **Simple**
  - Java is very easy to learn, and its syntax is simple, clean and easy to understand.
  - Java syntax is based on C++ (so easier for programmers to learn it after C++).
  - Java has removed many complicated and rarely-used features, for example, explicit pointers, operator overloading, etc.
  - There is no need to remove unreferenced objects because there is an Automatic Garbage Collection in Java.

Features of Java

- **Secured**
  - Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:
    - Java Programs run inside a virtual machine sandbox
    - No explicit pointer

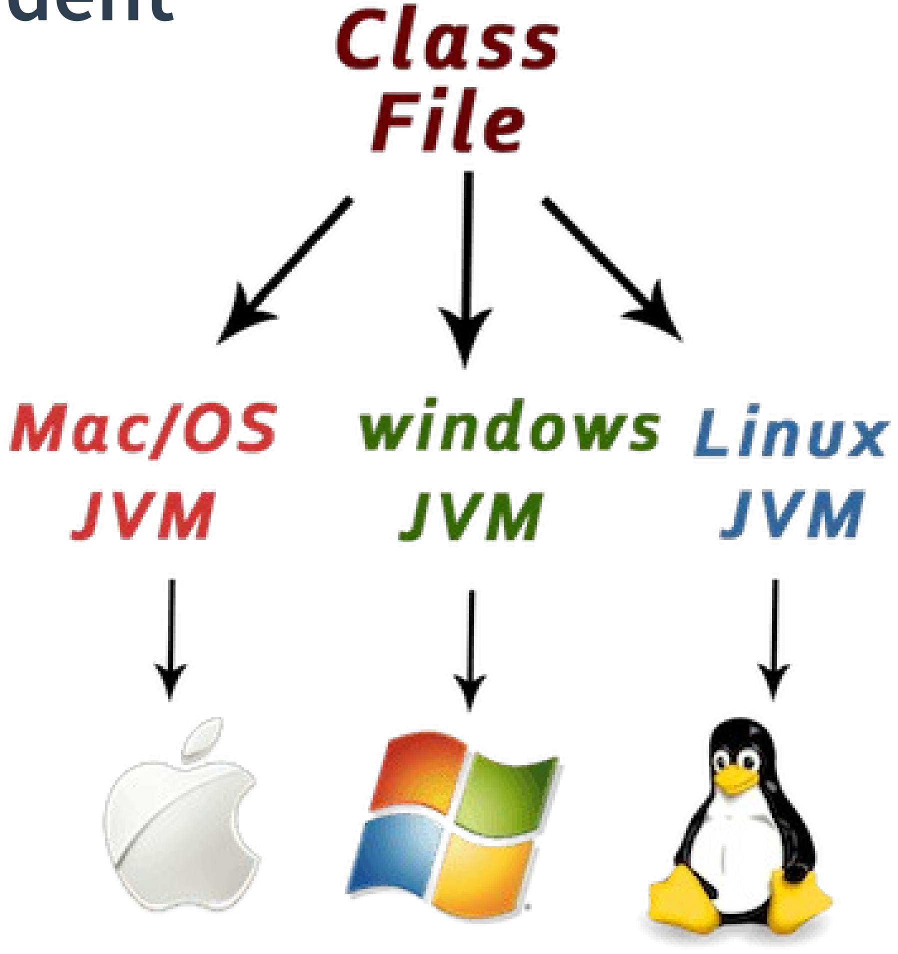

Uses Runtime Environment of operating system
C++ Application
OS
JAVA Application
Uses Runtime Environment of its own
JVM
OS

Features of Java

- Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language (WORA).
- The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:
  - Runtime Environment
  - API(Application Programming Interface)
- 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.

Features of Java

- Platform Independent



Features of Java

- **Architecture-neutral**
  - Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
  - In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.
- **Robust simply means strong, Java is robust because:**
  - It uses strong memory management.
  - There is a lack of pointers that avoids security problems.
  - There is automatic garbage collection in java which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
  - There are exception handling and the type checking mechanism in Java. All these points make Java robust.

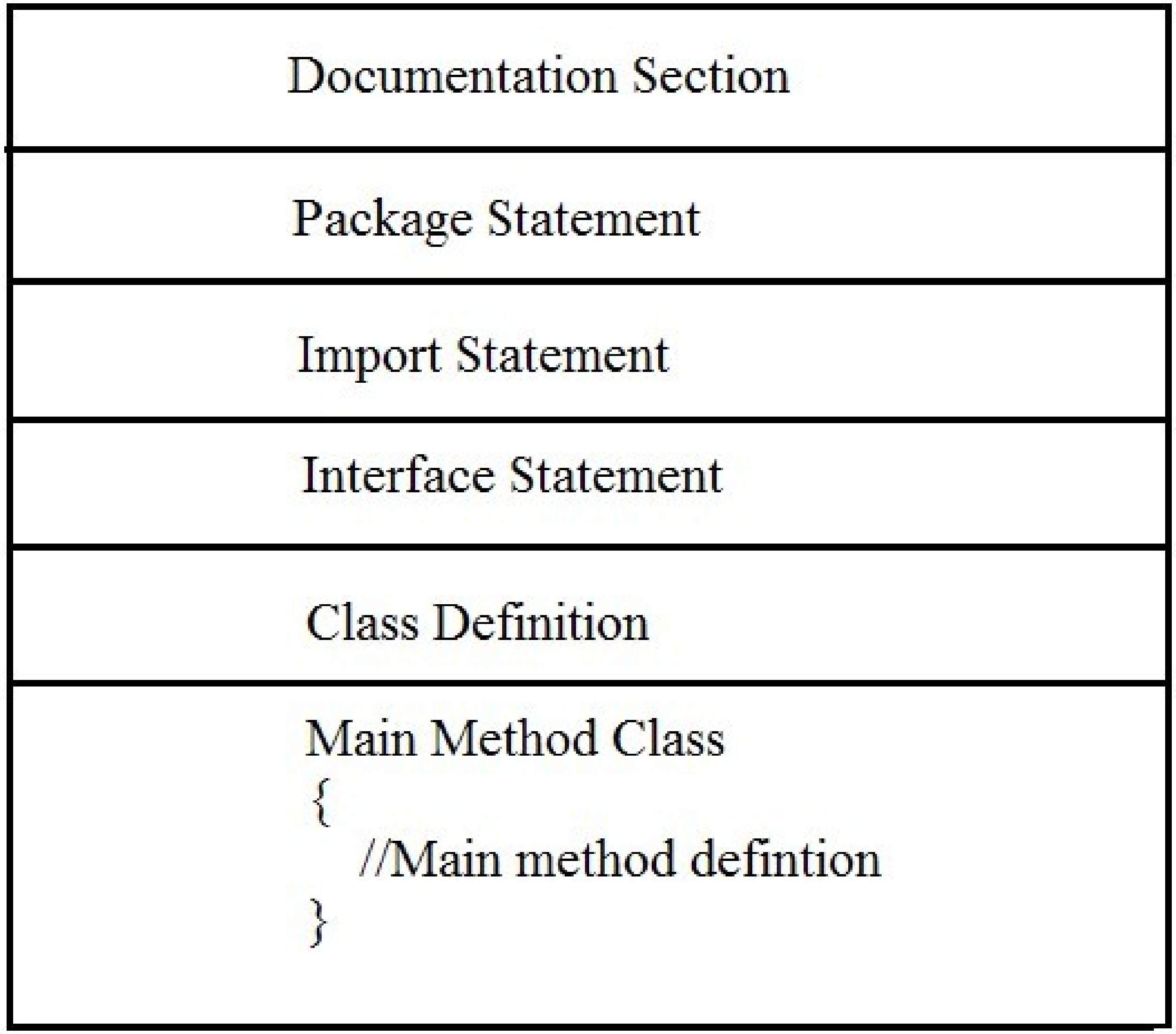
Features of Java

- **Distributed**
  - Java is distributed because it facilitates users to create distributed applications in Java. RMI and EJB are used for creating distributed applications. This feature of Java makes us able to access files by calling the methods from any machine on the internet.
- **Portable**
  - Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.
- **High-performance**
  - Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code. It is still a little bit slower than a compiled language (e.g., C++). Java is an interpreted language that is why it is slower than compiled languages, e.g., C, C++, etc.

Features of Java

- **Interpreted**
  - To run a Java program, we use the **Java** interpreter to execute the compiled byte-codes.
  - rapid turn-around development
  - Software author is protected, since binary byte streams are downloaded and not the source code
- **Multi-threaded**
  - A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.
- **Dynamic**
  - Java is a dynamic language. It supports dynamic loading of classes. It means classes are loaded on demand. It also supports functions from its native languages, i.e., C and C++.
  - Java supports dynamic compilation and automatic memory management (garbage collection).

Java Source File Structure



Documentation Section

Package Statement

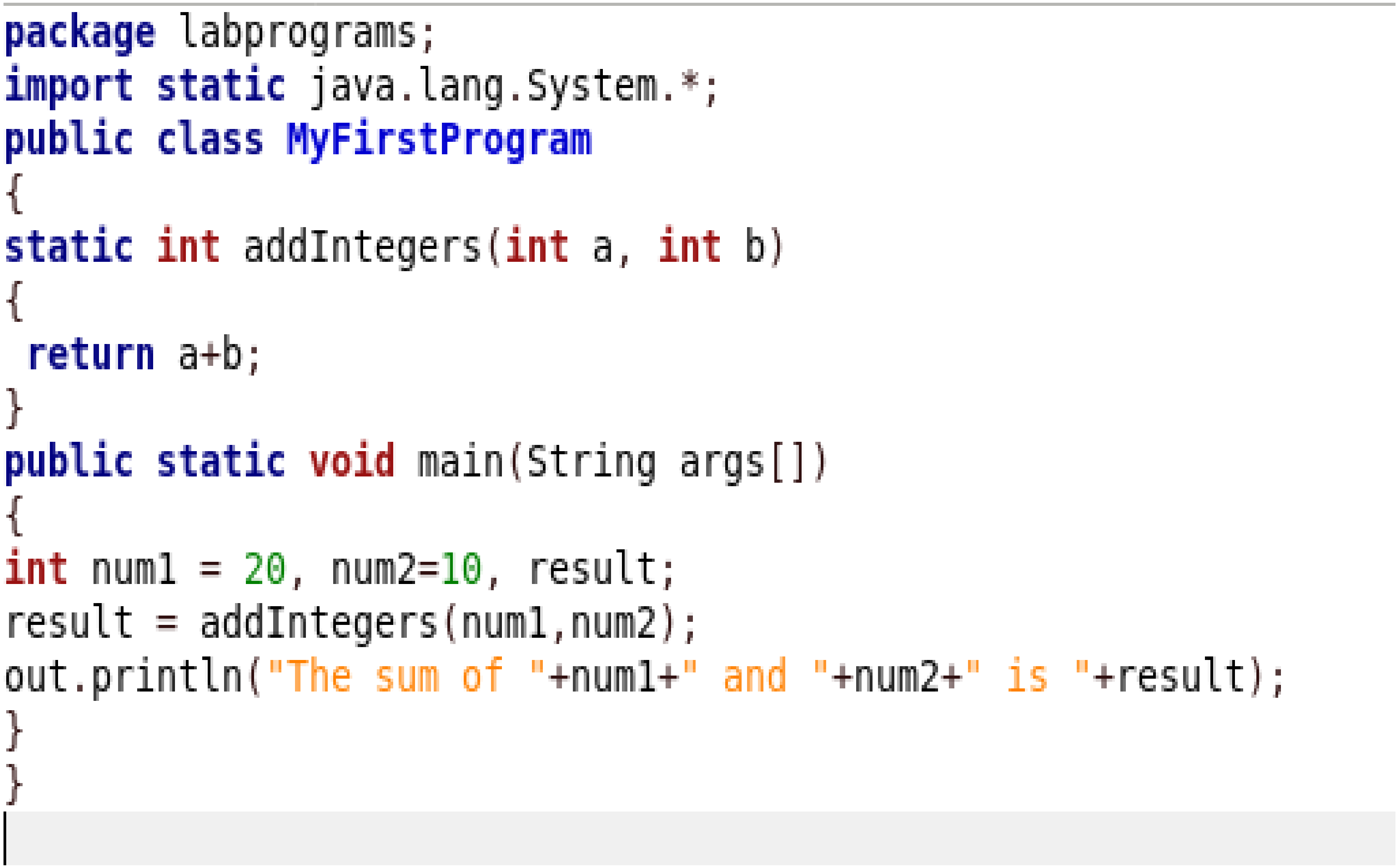
Import Statement

Interface Statement

Class Definition

```
Main Method Class
{
    //Main method defintion
}
```

Java Source File Structure



```
package labprograms;
import static java.lang.System.*;
public class MyFirstProgram
{
static int addIntegers(int a, int b)
{
 return a+b;
}
public static void main(String args[])
{
int num1 = 20, num2=10, result;
result = addIntegers(num1,num2);
out.println("The sum of "+num1+" and "+num2+" is "+result);
}
}
```

Java Source File Structure

- A java program can contain any no of classes, but atmost one class can be declared as public.
- If there is a public class then name of the class and the program name must be matched ,else CE will raise. If there is no public class, then we can use any name for the java program and there are no restrictions.

1)We can compile a java program but not java class (can execute).

2).class file generation is not based on name of program, whenever we are compling a java program for every class present in program a seperate .class file will be generated.

3)whenever we are executing a java class, the corresponding main method will be executed.

4)If .class doesnt contain main method then we will get runtime exception saying "NoSuchMethodError : main"

5)Whenever we are executing a java class,if .class is not available then we will get RE: "NoClassDefFoundError : name "

Basic programming constructs

- The obvious reason that object-oriented programming languages use objects is due to the power that design principles such as inheritance, information hiding and polymorphism provide the programmer.
- Even though languages may be object-oriented, most also still use the basic constructs of programming and algorithms developed in earlier programming languages. Java is no exception.

Basic programming constructs

- Programs are computer code that provide sequences of instructions - they can be large or small.
- Algorithms are the "recipes" ... the sequence of steps used to achieve the desired goal.
- All algorithms are made up of the following control constructs, which direct the flow of the program:
  - sequences (assignment statements, IO calls)
  - repetitions/loops (while, for, do)
  - decisions/selections (if/then, switch)
  - method invocation

Basic Data Types

- Types

– boolean	either true or false
– char	16 bit Unicode 1.1
– byte	8-bit integer (signed)
– short	16-bit integer (signed)
– int	32-bit integer (signed)
– long	64-bit integer (singed)
– float	32-bit floating point (IEEE 754-1985)
– double	64-bit floating point (IEEE 754-1985)

- String (class for manipulating strings)
- Java uses Unicode to represent characters internally(Link)

Variables

- Local Variables are declared within the block of code
- Variable has a type preceding the name
- Initial value is set by initialization expressions
  - type variableName = initialValue;
  - e.g. int x = 1;
- Variables can be defined just before their usage (unlike C)
  - e.g., for(int i = 0; i < 10; i+ +)

Constants

- Constants are similar to variables except that they hold a fixed value. They are also called "READ" only variables.
- Constants are declared with the reserved word "final".
  - final int MAX_MARK = 100;
  - final double PI = 3.1428;
- By convention upper case letters are used for defining constants.

Declaring Constants - example

```
class CircleArea
{
public static void main(String args[ ] )
{
final double PI = 3.1428;
double radius = 5.5; / / in cms
double area;
area = PI * radius * radius;
System.out.println("Circle Radius = "+ radius+ " Area= "+ area);
}
}
```

Comments

- **English text scattered through the code are comments**
- **JAVA supports 3 types of comments**

  / * * / - Usually used from multi-line comments

  //- Used for single line comments

  / * * * / - Documentation comments

Javadoc

- Effort to make Java self-documenting
- True OOP style, encapsulate documentation within code.
- Comments beginning with / * * and ending with * / can be extracted and turned into html documentation.
- Additional formatting using javadoc tags
  - @author, @see, @version, @param, @exception

Control Flow

- Control Flow Statements in JAVA
  - while loop
  - for loop
  - do-while loop
  - if-else statement
  - switch statement
- JAVA does not support a goto statement

Control Flow - Examples

```
while (squared <= MAX)
{
squared = lo * lo; // Calculate square
System.out.println(squared);
lo = lo + 1; /* Compute the new lo value */
}
```

Control Flow - Examples

```
for (int i = 1; i < MAX; i++)
{
System.out.println(i); // prints 1 2 3 4 5 ...
}
```

Control Flow - Examples

```
do
{
squared = lo * lo; // Calculate square
System.out.println(squared);
lo = lo + 1; /* Compute the new lo value */
} while (squared <= MAX);
```

Control Flow - Examples

```
if ( i < 10)
{
System.out.println(“i is less than 10” );
}
else
{
System.out.println(“i is greater than or equal to 10”);
}
```

Control Flow - Examples

```
switch (c)
{
case ‘a’:
System.out.println (“ The character is ‘a’” );
break;
case ‘b’;\:
System.out.println (“ The character is ‘b’” );
break;
default:
System.out.println (“ The character is not ‘a’ or ‘b’” );
break;
}
```

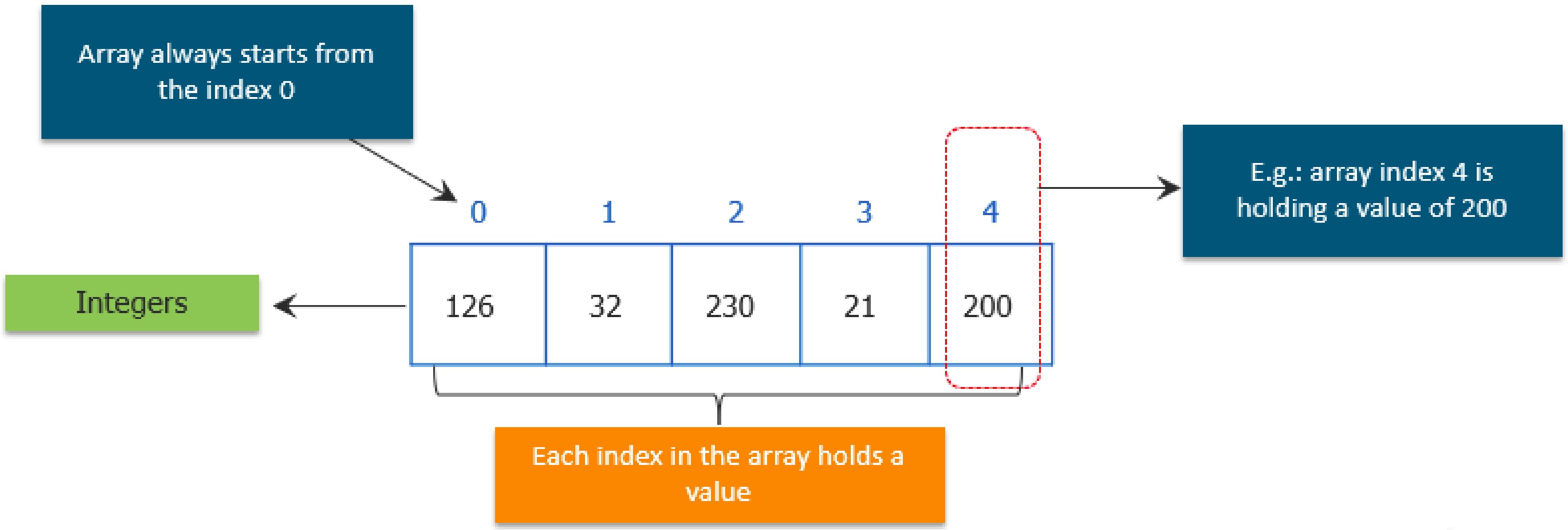
Arrays

- **Why exactly we need Java Array:**
  - Arrays are an important structure to hold data.
  - Java allows us to hold many objects of the same type using arrays.
  - It can be used with the help of a loop to access the elements by their index.

Arrays

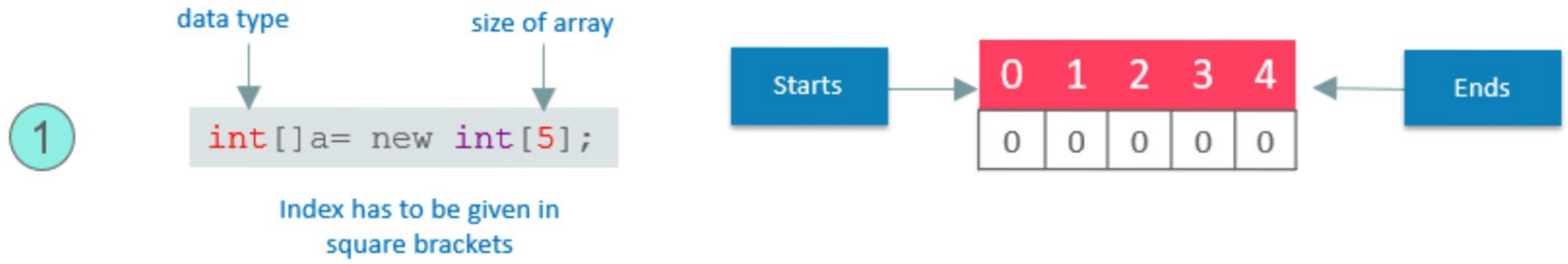
- Arrays in Java are homogeneous data structures implemented in Java as objects.
- Arrays store one or more values of a specific data type and provide indexed access to store the same.
- A specific element in an array is accessed by its index.
- Arrays offer a convenient means of grouping related information.

Arrays

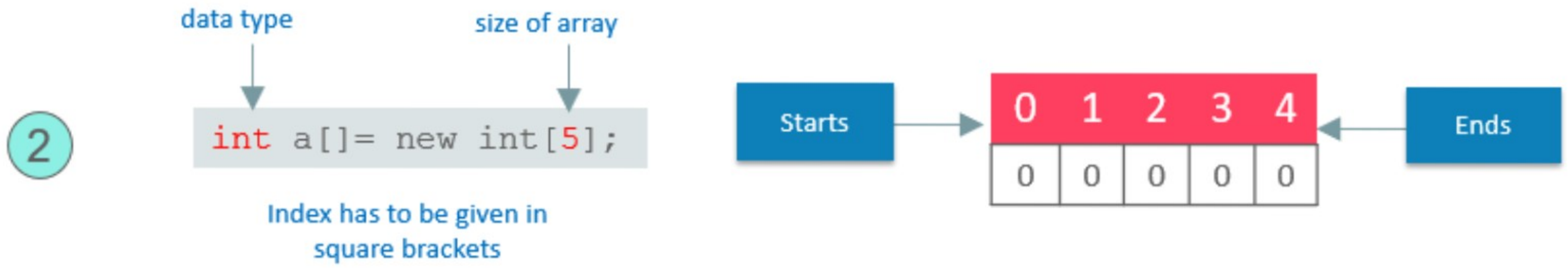

Array always starts from the index 0
0
1
2
3
4
E.g.: array index 4 is holding a value of 200
Integers
126
32
230
21
200
Each index in the array holds a value

- We can declare arrays in different ways.
  - type <var-name> [ ]
  - Example:- int month_days[ ];
  - type <var-name> = new type[size]

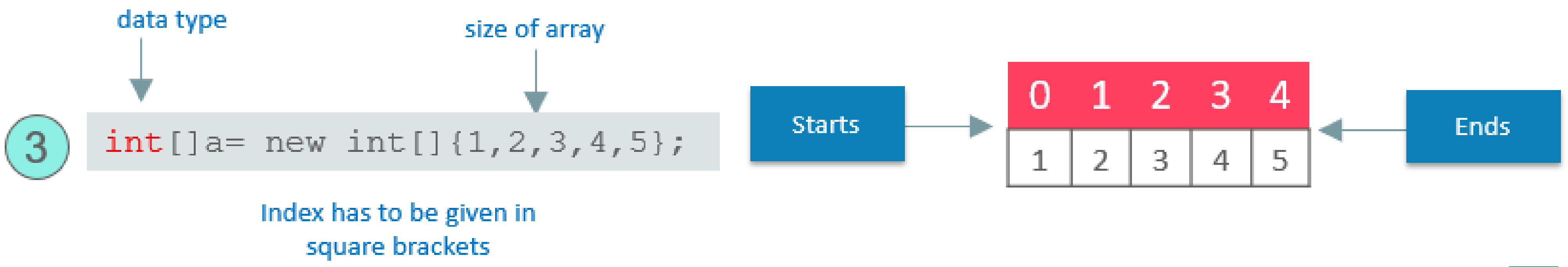
Arrays



```
int[]a= new int[5];
```



```
int a[]= new int[5];
```

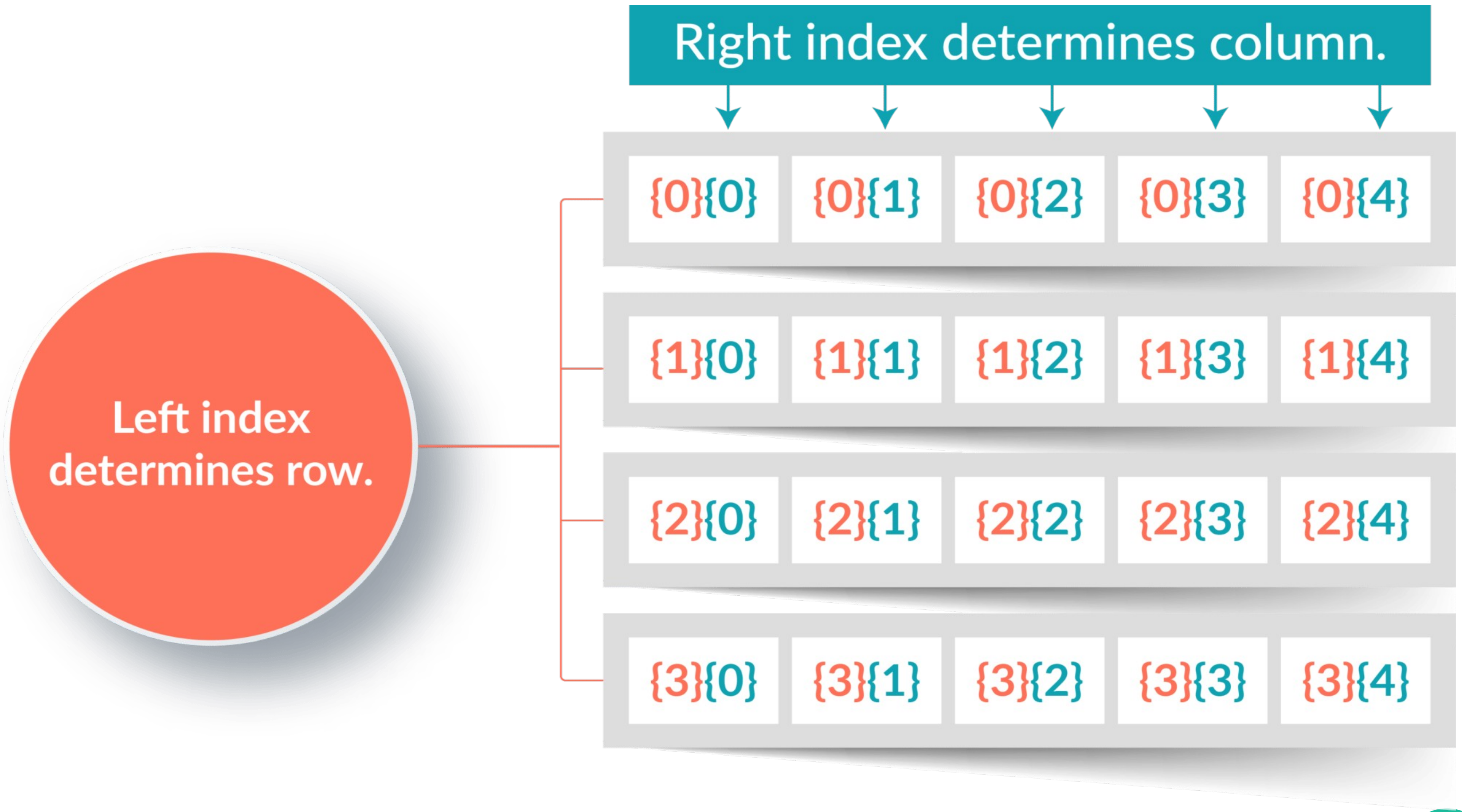


```
int[]a= new int[]{1,2,3,4,5};
```

Arrays

- **Multidimensional Array**
  - Multidimensional arrays are arrays of arrays.
  - To declare it, we have to specify each additional index using another set of square brackets.
  - Example int mul [ ][ ] = new int [4][5];

Arrays


Right index determines column.
Left index determines row.
{0}{0}
{0}{1}
{0}{2}
{0}{3}
{0}{4}
{1}{0}
{1}{1}
{1}{2}
{1}{3}
{1}{4}
{2}{0}
{2}{1}
{2}{2}
{2}{3}
{2}{4}
{3}{0}
{3}{1}
{3}{2}
{3}{3}
{3}{4}

Enhanced For-Loop

- The enhanced for-loop is a popular feature introduced with the Java SE platform in version 5.0.
- Its simple structure allows one to simplify code by presenting for-loops that visit each element of an array/collection without explicitly expressing how one goes from element to element.
- Because the old style of coding didn't become invalid with the new for-loop syntax, you don't have to use an enhanced for-loop when visiting each element of an array/ collection.

Enhanced For-Loop

```
for (int i=0; i < array.length; i++)
{
   System.out.println("Element: " + array[i]);
}
for (String element : array)
{
   System.out.println("Element: " + element);
}
```

Enhanced Vs Normal

Enhanced vs Normal

```
public class ForLoopDemo
{
  public static void main(String[] args)
  {
   int array[] = new int [] {1,2,3,4,5,6,7,8,9,10};
   long start = System.nanoTime();
   for (int i=0; i < array.length; i++)  { System.out.print("Element: " + array[i]+"\t");}
   long end = System.nanoTime();;
   System.out.println("Counting takes " + (end - start)/1000 + " micro seconds");
   start = System.nanoTime();
   for (int element : array){ System.out.print("Element: " + element+"\t");}
   end = System.nanoTime();;
   System.out.println("Counting takes " + (end - start)/1000 + " micro seconds");
  }
}
```

String

- String is a sequence of characters, for e.g. “Hello” is a string of 5 characters.
- In java, string is an immutable object which means it is constant and cannot be changed once it has been created.
- Creating a String
  - String literal
  - Using new keyword

String

- String literal
  - String str1 = "Welcome";
  - String str2 = "Welcome";
- String is an object in Java. However we have not created any string object using new keyword above.
- The compiler does that task for us it creates a string object having the string literal and assigns it to the provided string instances.

String

- But if the object already exist in the memory it does not create a new Object rather it assigns the same old object to the new instance, that means even though we have two string instances above(str1 and str2) compiler only created one string object (having the value "Welcome") and assigned the same to both the instances. Example
- if we want to have two different object with the same string? For that we would need to create strings using new keyword.

String

- To overcome that approach we can create strings like this:
  - String str1 = new String("Welcome");
  - String str2 = new String("Welcome");

Java String Methods

- char charAt(int index): It returns the character at the specified index.
- Specified index value should be between 0 to length() -1 both inclusive.
- It throws IndexOutOfBoundsException if index<0|| >= length of String.

Java String Methods

- boolean equals(Object obj): Compares the string with the specified string and returns true if both matches else false.
- boolean equalsIgnoreCase(String string): It works same as equals method but it doesn’t consider the case while comparing strings. It does a case insensitive comparison.

Java String Methods

- int compareTo(String string): This method compares the two strings based on the Unicode value of each character in the strings.

- int compareToIgnoreCase(String string): Same as CompareTo method however it ignores the case during comparison.

Java String Methods

- boolean startsWith(String prefix, int offset): It checks whether the substring (starting from the specified offset index) is having the specified prefix or not.
- boolean startsWith(String prefix): It tests whether the string is having specified prefix, if yes then it returns true else false.
- boolean endsWith(String suffix): Checks whether the string ends with the specified suffix.

Java String Methods

- int hashCode(): It returns the hash code of the string.
- int indexOf(int ch): Returns the index of first occurrence of the specified character ch in the string.
- int indexOf(int ch, int fromIndex): Same as indexOf method however it starts searching in the string from the specified fromIndex.
- int lastIndexOf(int ch): It returns the last occurrence of the character ch in the string.

Java String Methods

- int lastIndexOf(int ch, int fromIndex): Same as lastIndexOf(int ch) method, it starts search from fromIndex.
- int indexOf(String str): This method returns the index of first occurrence of specified substring str.
- int lastindexOf(String str): Returns the index of last occurrence of string str.

Java String Methods

- String substring(int beginIndex): It returns the substring of the string. The substring starts with the character at the specified index.
- String substring(int beginIndex, int endIndex): Returns the substring. The substring starts with character at beginIndex and ends with the character at endIndex.
- String concat(String str): Concatenates the specified string “str” at the end of the string.
- String replace(char oldChar, char newChar): It returns the new updated string after changing all the occurrences of oldChar with the newChar.

Java String Methods

- boolean contains(CharSequence s): It checks whether the string contains the specified sequence of char values. If yes then it returns true else false. It throws NullPointerException of ‘s’ is null.
- String toUpperCase(Locale locale): Converts the string to upper case string using the rules defined by specified locale.
- String toUpperCase(): Equivalent to toUpperCase(Locale.getDefault()).
- public String intern(): This method searches the specified string in the memory pool and if it is found then it returns the reference of it, else it allocates the memory space to the specified string and assign the reference to it.
- public boolean isEmpty(): This method returns true if the given string has 0 length. If the length of the specified Java String is non-zero then it returns false.

Java String Methods

- public static String join(): This method joins the given strings using the specified delimiter and returns the concatenated Java String
- String replaceFirst(String regex, String replacement): It replaces the first occurrence of substring that fits the given regular expression “regex” with the specified replacement string.
- String replaceAll(String regex, String replacement): It replaces all the occurrences of substrings that fits the regular expression regex with the replacement string.

Java String Methods

- String[] split(String regex, int limit): It splits the string and returns the array of substrings that matches the given regular expression. limit is a result threshold here.
- String[] split(String regex): Same as split(String regex, int limit) method however it does not have any threshold limit.
- String toLowerCase(Locale locale): It converts the string to lower case string using the rules defined by given locale.
- public static String format(): This method returns a formatted java String

Java String Methods

- String toLowerCase(): Equivalent to toLowerCase(Locale. getDefault()).
- String trim(): Returns the substring after omitting leading and trailing white spaces from the original string.
- char[] toCharArray(): Converts the string to a character array.
- static String copyValueOf(char[] data): It returns a string that contains the characters of the specified character array.
- static String copyValueOf(char[] data, int offset, int count): Same as above method with two extra arguments – initial offset of subarray and length of subarray

Java String Methods

- void getChars(int srcBegin, int srcEnd, char[] dest, int destBegin): It copies the characters of src array to the dest array. Only the specified range is being copied(srcBegin to srcEnd) to the dest subarray(starting fromdestBegin).
- static String valueOf(): This method returns a string representation of passed arguments such as int, long, float, double, char and char array.
- boolean contentEquals(StringBuffer sb): It compares the string to the specified string buffer.
- boolean regionMatches(int srcoffset, String dest, int destoffset, int len): It compares the substring of input to the substring of specified string.

Java String Methods

- boolean regionMatches(boolean ignoreCase, int srcoffset, String dest, int destoffset, int len): Another variation of regionMatches method with the extra boolean argument to specify whether the comparison is case sensitive or case insensitive.
- byte[] getBytes(String charsetName): It converts the String into sequence of bytes using the specified charset encoding and returns the array of resulted bytes.
- byte[] getBytes(): This method is similar to the above method it just uses the default charset encoding for converting the string into sequence of bytes.
- int length(): It returns the length of a String.
- boolean matches(String regex): It checks whether the String is matching with the specified regular expression regex.

String is Immutable in Java

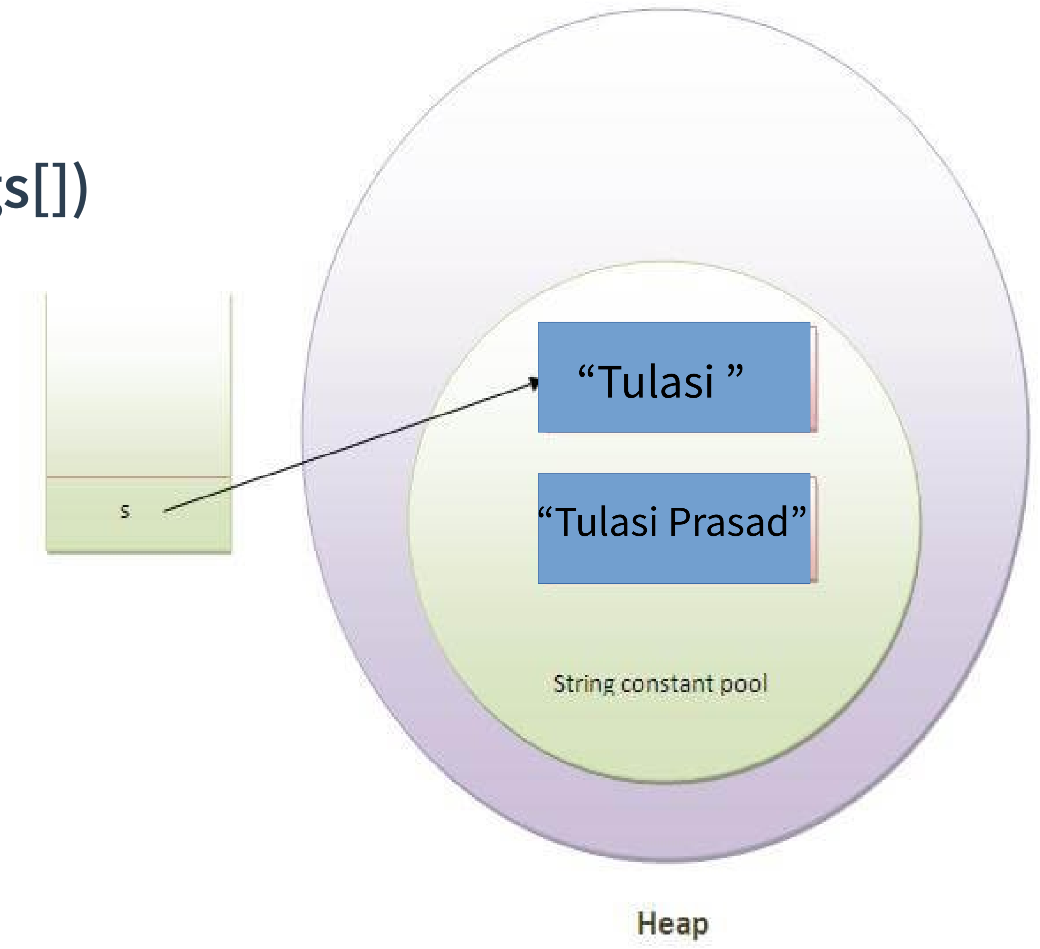
- **In java, string objects are immutable. Immutable simply means un-modifiable or unchangeable.**
- **Once string object is created its data or state can't be changed but a new string object is created.**
- **Why string objects are immutable in java?**
  - Because java uses the concept of string literal. Suppose there are 5 reference variables,all refers to one object.
  - If one reference variable changes the value of the object, it will be affected to all the reference variables.
  - That is why string objects are immutable in java.

String is Immutable in Java

- When we create a String using double quotes, it first looks for the String with same value in the JVM string pool, if found it returns the reference else it creates the String object and then place it in the String pool.
- This way JVM saves a lot of space by using same String in different threads. But if new operator is used, it explicitly creates a new String in the heap memory.
- Since String is immutable in java, whenever we do String manipulation like concat, substring etc, it generates a new String and discard the older String for garbage collection.

String is Immutable in Java

```
class StringImmutable
{
 public static void main(String args[])
 {
  String s="Tulasi";
  String s1 = s.concat(" Prasad");
  System.out.println(s);
  System.out.println(s1);
 }
}
```

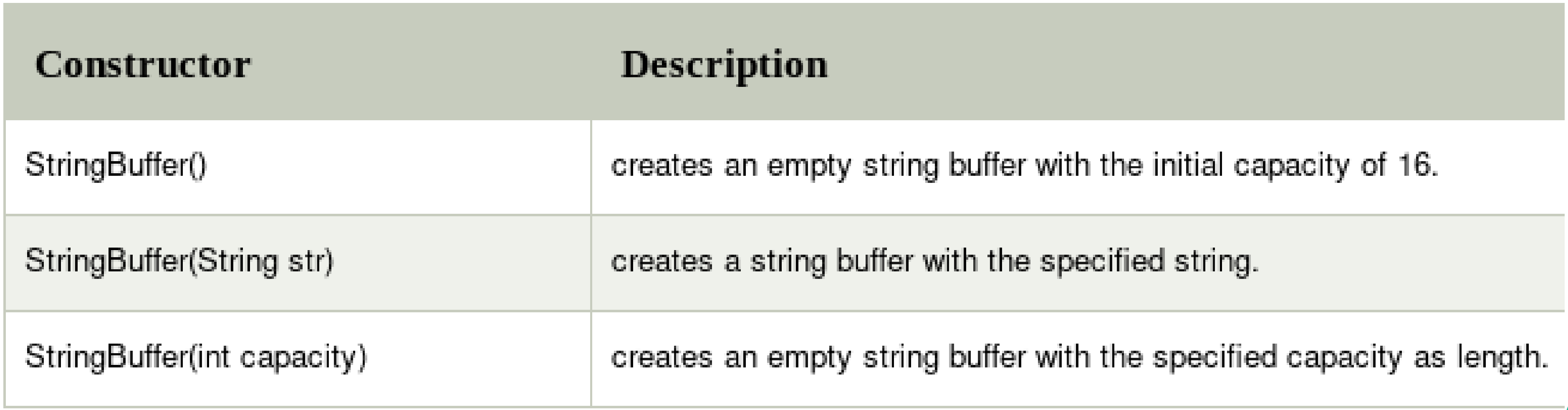


StringBuffer and StringBuilder

- String manipulations are heavy operations and generate a lot of garbage in heap.
- So Java has provided StringBuffer and StringBuilder class that should be used for String manipulation.
- StringBuffer and StringBuilder are mutable objects in java and provide append(), insert(), delete() and substring() methods for String manipulation.

StringBuffer

- StringBuffer class is used to create mutable (modifiable) string.
- The StringBuffer class in java is same as String class except it is mutable i.e. it can be changed.
- Important Constructors of StringBuffer class



Constructor	Description
StringBuffer()	creates an empty string buffer with the initial capacity of 16.
StringBuffer(String str)	creates a string buffer with the specified string.
StringBuffer(int capacity)	creates an empty string buffer with the specified capacity as length.

StringBuffer Methods

- length(): Returns the StringBuffer object's length.
- capacity(): Returns the capacity of the StringBuffer object.
- append(): appends the specified argument string representation at the end of the existing String Buffer.
- insert(): insert() method takes two parameters – the index integer value to insert a value and the value to be inserted. The index tells StringBuffer where to insert the passed character sequence. Again this method is overloaded to work with primitive data types and Objects

StringBuffer Methods

- reverse(): Reverses the existing String or character sequence content in the buffer and returns it. The object must have an existing content or else a NullPointerException is thrown.
- delete(int startIndex, int endIndex): accepts two integer arguments. The former serves as the starting delete index and latter as the ending delete index. Therefore the character sequence between startIndex and endIndex–1 are deleted. The remaining String content in the buffer is returned.

StringBuilder

- Java StringBuilder class is mutable sequence of characters. StringBuilder Class can be comparable to String however the StringBuilder class provides more versatility because of its modification features.
- StringBuilder class provides an API similar to StringBuffer, but unlike StringBuffer, it doesn’t guarantee thread safety.
- Java StringBuilder class is designed for use as a drop-in replacement for StringBuffer in places where the string buffer was being used by a single thread (as is generally the case).
- If execution speed and performance is a factor, StringBuilder class can be used in place of StringBuffer.

StringBuffer vs StringBuilder

StringBuffer	StringBuilder
Synchronized, hence thread safe.	Not synchronized, not thread safe.
Operates slower due to thread safety feature	Better performance compared to StringBuffer
Has some extra methods – substring, length, capacity etc.	Not needed because these methods are present in String too
Introduced in Java 1.2	Introduced in Java 1.5 for better performance.

String vs StringBuffer vs StringBuilder

- String is immutable whereas StringBuffer and StringBuider are mutable classes.
- StringBuffer is thread safe and synchronized whereas StringBuilder is not, thats why StringBuilder is more faster than StringBuffer.
- String concat + operator internally uses StringBuffer or StringBuilder class.
- For String manipulations in non-multi threaded environment, we should use StringBuilder else use StringBuffer class.

Wrapper Classes

- In the OOPs concepts guide, we learned that object oriented programming is all about objects.
- The eight primitive data types byte, short, int, long, float, double, char and boolean are not objects, Wrapper classes are used for converting primitive data types into objects, like int to Integer etc.

Wrapper Classes

Primitive	**Wrapper class**
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Why we need wrapper class in Java

- The primitive data types are not objects so they do not belong to any class. While storing in data structures which support only objects, it is required to convert the primitive type to object first which we can do by using wrapper classes.
- Wrapper objects hold much more memory compared to primitive types. So use primitive types when you need efficiency and use wrapper class when you need objects instead of primitive types.
- Wrapper class objects allow null values while primitive data type doesn't allow it.

Wrapper Classes

Converting a primitive type to Wrapper object

```
public class JavaExample{
  public static void main(String args[])
{
int num=100;
Integer obj=Integer.valueOf(num);
System.out.println(num+ " "+ obj);
  }
}
```

Wrapper Classes

```
public class JavaExample{
  public static void main(String args[]){
//Creating Wrapper class object
Integer obj = new Integer(100);
//Converting the wrapper object to primitive
int num = obj.intValue();
System.out.println(num+ " "+ obj);
  }
}
```

References

- https://docs.oracle.com/javase/tutorial/
- https://www.javatpoint.com/features-of-java
- http://www.java2novice.com/java-fundamentals/
- http://archive.oreilly.com/oreillyschool/courses/java2/
- https://docs.oracle.com/javase/7/docs/api/
- https://www.javatpoint.com/
- https://beginnersbook.com/java-tutorial-for-beginners-with-examples/

Thank You!