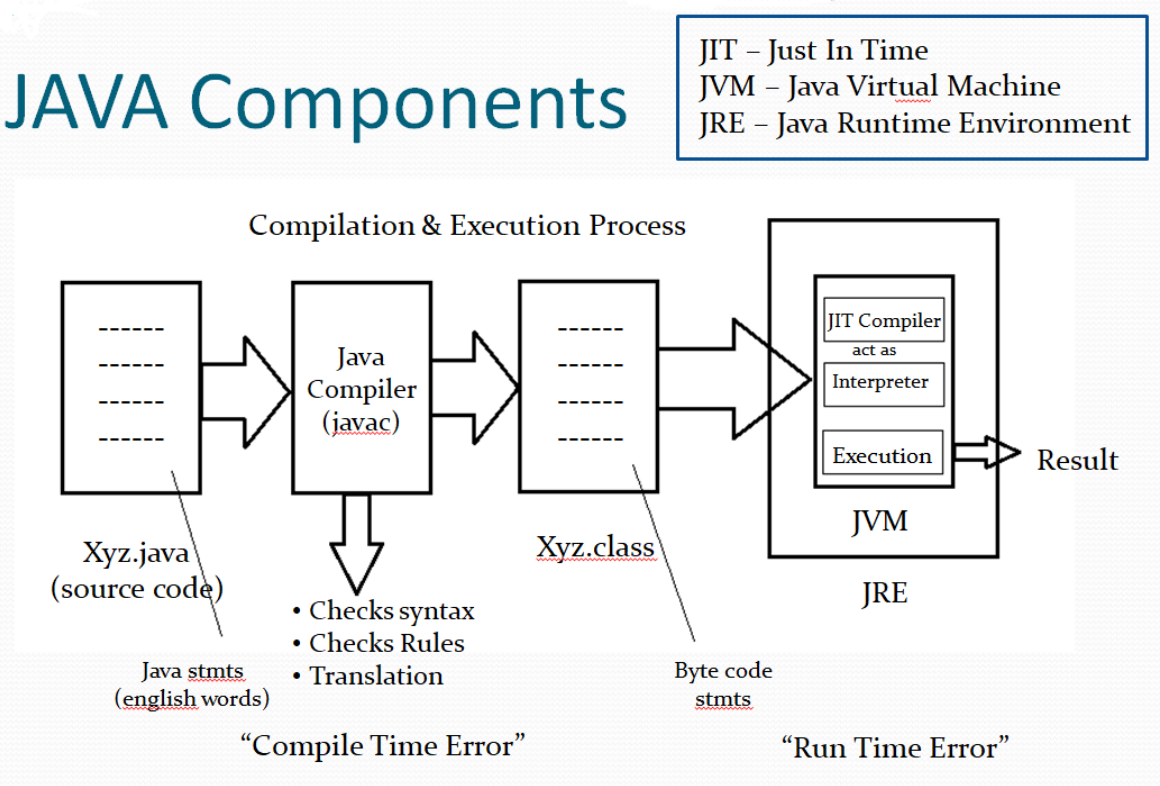
What is JAVA...?

- ✓ Java is a general purpose, **high-level programming language** developed by **Sun Microsystems.**
- ✓ A small team of engineers, known as the *Green Team*, initiated the language in 1991.
- ✓ Java was originally developed by **James Gosling.**
- ✓ Java was originally called ***OAK,*** and was designed for handheld devices and set-top boxes.
- ✓ Oak was unsuccessful, so in 1995 Sun changed the name to Java and modified the language.



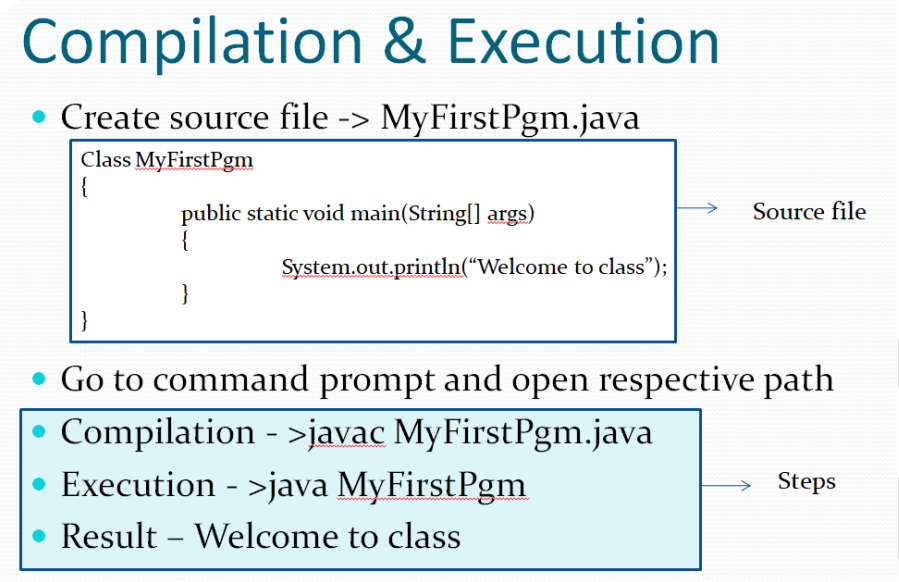
- ✓ Developing a pgm in Java language includes 3 steps
  - Source code creation
  - Source code compilation
  - Execution
- ✓ The source code creation is done by writing Java pgm using the syntax of Java language.
- ✓ The source code should be saved with the extension of **.java**
- ✓ We can create the source code using any text editor or IDE tools.
- ✓ The source code is compiled to get executable format.
- ✓ Java compiler is used to do the compilation of source code.

- ✓ Java compiler checks the syntax & rules before compiling it.
- ✓ If any syntax or rules mistakes, then compiler throws an error called "**Compiler error**".
- ✓ The java compiler translates the Java stmts into **Byte Codes.** The byte codes are saved in the files with the extension **.class**
- ✓ The execution of the pgm is done by JVM, inside JVM the JIT compiler compiles the byte codes to the m/c level formats & it is executed by JVM.
- ✓ The JRE is responsible to provide the necessary environment to the JVM. So that the JVM can executes the byte codes.
- ✓ The .class file is OS independent but JRE dependent. We can run the .class files on any OS provided the JRE's available in that system. This is known as "**Platform Independent**".
- ✓ The Java s/w is released in "**Development Tool Kit**" known as **JDK** which contains the necessary development tools like **Java compiler, JRE & other libraries.**
- ✓ There are 2 types of JRE
    - Public JRE – used whenever the java pgms run on **Server**.
    - Private JRE – used whenever the java pgms run on **local m/c.**

Features of Java

- ✓ **Simple** - confusing features in C++ are removed in Java like pointers etc..
- ✓ **Secure** - provides data security through encapsulation.
    - Programmes run within the JVM which protects from unauthorized access to system resources.
- ✓ **Portable** - Bytecode helps Java to achieve portability.
- ✓ **Object – Oriented** - it supports all the features of object oriented model like: Encapsulation, Inheritance Polymorphism & Abstraction.
- ✓ **Robust** - Type checking & Exception handling helps to make the programs robust.
- ✓ **Multithreaded** - supports multithreading which is not supported by C and C++.
- ✓ **Architecture neutral** - Since Java applications can run on any kind of CPU, Java is architecture – neutral.
- ✓ **Interpreted & High Performance** - JIT compiler converts the byte code into machine code piece by piece and caches them for future use. This enhances the program performance means it executes rapidly.

- ✓ **Distributed** - supports distributed computation using Remote Method Invocation (RMI) concept.
- ✓ **Dynamic** - The Java Virtual Machine (JVM) maintains a lot of runtime information about the program and the objects in the program.
  - Libraries are dynamically linked during runtime.



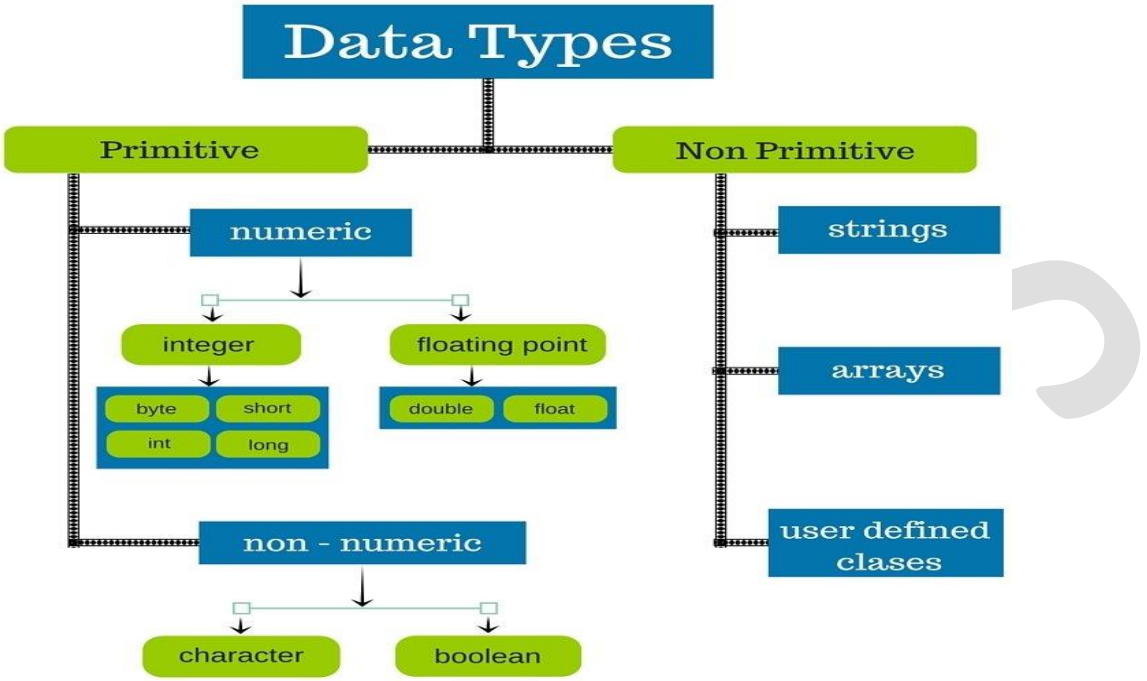
Java Operators

Types	Operators
Arithmetic Operators	+, -, *, /, %, ++ , --
Relational Operators	==, !=, >, <, >=, <=
Bitwise Operators	&, |, ^, ~, <<, >>, >>>
Logical Operators	&&, ||, !
Assignment Operators	=, +=, -=, *=, /=, %=, <<=, >>=, &=, ^=, |=
Misc Operators	? :, instanceof, new, .(dot)
Unary Operators	++, --

"+" Operator

- ✓ Java supports operator overloading concept only for + Operator.
  - Addition of numbers
    - 20 + 40 = 60

- Concatenation of Strings
  - "java" + "developer" = javadeveloper
  - "java"+ 10 = java10



Keywords

- ✓ Keywords are predefined preserved word which is used for particular purpose.
- ✓ Each keyword has its own meaning & user cannot modify the meaning.
- ✓ The programmer can built the programme by using keywords.
- ✓ In java language all the keywords are represented in "lower case".

Identifiers

- ✓ Identifiers are used to represent a value in the programme.
- ✓ While using the identifiers we should follow the below rules
  - An identifier can be Alpha-Numeric characters.
  - All identifier should begin with Alphabets only. If it begins with numeric compiler throws error.
  - Special character "_" and "$" can be used.
  - It should not have any space.

Valid identifiers	Invalid identifiers
empid	123empid
empid123	emp id
emp_id	emp@id

Control Statements

- ✓ Control statements control the order of execution in a program, based on data values and conditional logic.
- ✓ Types
  - If statement
  - If – else statement
  - If – else ladder statement
  - Nested if statement
  - Switch statement
- ✓ These are also known as **selection statements**.

Looping Statements

- ✓ Looping statements repeat a specified block of code until a given condition is met.
- ✓ **Types**
  - While loop
  - Do – while loop
  - For loop

Functions/Methods

- ✓ Functions are used to defined the operation or the task in a program.
- ✓ By developing functions we can achieve modularity & code reusability.
- ✓ While developing a pgm each task is built or coded using the function
- ✓ The syntax of declaring & defining a function is :

```
<access specifiers> <modifiers> returntype function_name(<argument lists>)
{
        ----------
        ----------
        return value;
}
```

- ✓ The function arguments are used to pass values to the function body.
- ✓ The function arguments should be declared in the function declaration line.
- ✓ We can declare function without argument or with argument.
- ✓ We can declare function with multiple arguments, multiple arguments should be separated by comma(,)
- ✓ The function argument is local to the function body.

Function Returntype

- ✓ The function return type specifies the type of value returning by the functions.
- ✓ We should specify the data type in the return type field.
- ✓ A function can return a value by using "return" keyword.
- ✓ If a function doesn't want to return a value, then in the return type we should mention "void".
- ✓ A function can return only one value at a time.

How to read inputs from keyboard

- ✓ Step 1: improt java.util.Scanner;
- ✓ Step 2: create Scanner class object
  - Scanner scn = new Scanner(System.in);
- ✓ Step 3: Use funtions to read inputs
  - To read int value from keyboard
  - int x = scn.nextInt();
  - To read String value from keyboard
  - String st = scn.next();
  - To read double value from keyboard
  - double y = scn.nextDouble();

Arrays

- ✓ Declaration & Initialisation
  - Method 1: By specifying size

```
datatype[] referenceVariable = new datatype[size];
```

referenceVariable[0] = value 1;

. . .

referenceVariable[size-1] = value n;

- Method 2: Using array initializer

datatype[] referenceVariable = {value 1, value 2, . . . , value n};

❖ Class & Object

- ✓ A class[logical entity] is a definition block which defines the state & behaviour of the object.
- ✓ An entity which has its own states & behaviour is known as object[physical entity].
- ✓ The state represents the characteristics of object whereas behaviour represents the action or the functionality of the object.
- ✓ The object doesn't exist without a class definition.
- ✓ We can create any number of instances from a class each instance differs in the values of the state but same in behaviour.
- ✓ Defining anything in the body of a class is known as members of the class.
- ✓ The class body can contain member variable or member functions.
- ✓ Member variables are the variables declared & initialize in the body of the class where as the member function are the methods defined in the body of the class.
- ✓ The member variable are used to represent the data where as member function are used to represent the operation performed on the data.

❖ Member types

- ✓ We can define 2 types of members in a class body.
- ✓ static member types
- ✓ non – static member types
- ✓ The static member types are declared using static keyword it is also known as a class member because it is associated to the class.
- ✓ The static members of the class can be accessed in any class body by using the syntax.

  className.memberName
- ✓ The static members of a class are loaded one copy in the memory which can be modified.
- ✓ The non-static members are defined without static keyword.
- ✓ It is associated to the instance of the class, hence it is also known as **Instance member**.
- ✓ To access the non-static members of a class we should create the instance of the class.

- ✓ The instance is created by using **new operator**.
- ✓ Whenever we create an instance of a class, a copy of the class gets loaded multiple copies in the memory.
- ✓ We can create 'n' no. of instance.
- ✓ The non-static members are loaded multiple copies in the memory.
- ✓ To access each instance we should create reference variable.
- ✓ The reference variable are used to identify the instance & to access the instance variables & instance functions.
- ✓ The reference variable are declared by using class type & it is non-primitive variables.
- ✓ Changes made to the instance variable of a instance will not reflect in other instances.
- ✓ An instance can be referred by any number of reference variables. In such case, if we change the instance property using one reference will be reflects in other references.

❖ NOTE

- ✓ A reference variable should always hold the address where the instances created in the memory. If not the reference variable should hold **null** or in other word a reference variable should point to instance if not points to null.
- ✓ If a reference variable is pointing to null & if we perform any operation using the reference then **JVM** throws **NullPointerException.**

❖ Constructor

- ✓ Constructor is a special member of a class used to provide the initialization to the instance variables of the class at the time of object creation.
- ✓ Every class defined in Java language should specified the constructor.
- ✓ The constructor can be defined either by compiler or by user.
- ✓ Based on who defines, constructor has 2 types
    1. Compiler defined constructor
    2. User defined constructor

1. Compiler defined constructor

- ✓ This will not have any parameter & it is known as **default constructor**.
- ✓ The compiler defines the constructor only when the class is not having any user defined constructor.

2. User defined constructor

- ✓ The constructor defined by user is known as user defined constructor.
- ✓ It is of 2 types
    1. Constructor without argument (No argument constructor)

2. Constructor with argument (Parameterized constructor)

- ✓ A constructor defined with parameterized constructor is known as a parameterized constructor.
- ✓ While defining a constructor the constructor name must be same has class name.
- ✓ The constructor should not specify any return type.

```
class Demo{
        int x;
        Demo(){
                x = 10;
        }
}
```

- ✓ If the constructor is defined with a return type then it will be treated as member function.
- ✓ If class has blocks & constructors then JVM executes blocks first & then constructor of the class.
- ✓ A constructor cannot be declared as **static.**
- ✓ In a class we can define any number of constructors provided the argument types of the constructor should vary, such constructors are known as **Overloaded constructors.**
- ✓ In other words, defining multiple constructors with different argument list is known as **Constructor Overloading.**
- ✓ We cannot define two constructors with same argument types.

❖ Need for Overloaded Constructor

- ✓ The overloaded constructors will help to create objects with different initialization of instance variable.

❖ "this" Keyword

- ✓ In Parameterized constructor to provide readability we make use "this" keyword.
- ✓ "this" keyword holds the address of current object.
- ✓ It must be used with Non-Static members.
- ✓ It is used to differentiate between Member variable/Class level variable & Local variables.

❖ Encapsulation

- ✓ Binding or wrapping the statements into blocks is called as Encapsulation.
- ✓ i.e., Members/Statements are wrapped into class body or method body

❖ Access Specifiers

- ✓ In order to restrict the access of members java provides 4 different access Specifiers.
- ✓ Namely:
  - o private
  - o package level/default
  - o protected
  - o public

❖ private

- ✓ private is the very high secured access specifier among all the access specifiers.
- ✓ private members can be accessed only with in the body of specific class body.
- ✓ i.e., this members can possible to access by the same class members that too with in the body itself.
- ✓ If these members are trying to access from outside the body then compiler throws error.

❖ package level/default

- ✓ This is the default access specifier provided to the member of a class.
- ✓ If member is not declared any access specifier then it will be considered as package level.
- ✓ This members can be access upto package level.

❖ protected

- ✓ This access specifier is also having access level upto its package.
- ✓ Addition to this, it can also access by the subclass of it from other package.
- ✓ (class(pack1) must hava "is-a" relationship with another class(pack2))

❖ public

- ✓ This access specifier is access by all the packages.

❖ Note

- ✓ Package is nothing but folder/directory.
- ✓ Inorder to access other package members we need to import the perticular class by specifying fully qualified class name
- ✓ fully qualified class name
  - o packageName.className
  - o ex: com.jsp.pack1.Demo
- ✓ Here upto pack1, it is package name & it should be in smaller case as per the industrial standards.
- ✓ Since class name begins with upper case we can easily differentiate between class & package name.

❖ Inheritance

- ✓ A class inheriting the properties from another class is known as "**Inheritance**".
- ✓ A class can inherits the properties by using "**extends**" keyword.
- ✓ The class from where members are inherited is known as **super class.**
- ✓ The class to which members are inherited is known as **sub class.**
- ✓ The sub class will always have the properties of super class.
- ✓ Sub class inherits only the **non-private properties** of super class.
- ✓ The following members are cannot be inherited by sub classes.
  - o private members
  - o Constructors
- ✓ A class can be declared as final, final classes cannot be inherited to subclass. In other words a class cannot extends from final class. We can create the instances of final class but we cannot extend it.
- ✓ We can also say that a class can have a **"has-a"** relationship with final class but cannot have **"is-a"** relationship.

❖ Type of inheritance

- ✓ **Single Inheritance –** In this type of inheritance a class inherits from on super class.
- ✓ **Multilevel Inheritance** – In this type, a class inherits from other class which is a subclass of some other class.
- ✓ **Multiple Inheritance** – In this type, a class inherits from more than one super class. Java doesn't support multiple inheritances.
- ✓ **Hierarchical Inheritance** – In this type, a super class will have more than one subclass. All the sub classes will have the same properties of super class.

❖ Why multiple inheritance doesn't supported in Java?

- ✓ The subclass constructor cannot make a call to **more** than **one** super class constructor.[Related to Constructor calling stmts]
- ✓ It leads to ambiguity of "**Diamond Problem**".

❖ Advantages of Inheritance

- ✓ Code reusability
- ✓ Software extensibility
- ✓ Modifiability

❖ this & super keyword

- ✓ **"this**" keyword is used refer the **current object** members.
- ✓ The reference on which the member is invoked is known as **current object.**
- ✓ "**this**" keyword must be **used** only in the **non-static** method context & **constructor** body. It **cannot** be used in **static context**.
- ✓ Java language provides a keyword "**super**" to access the super class non-static properties in the subclass methods.
- ✓ **super** keyword must be used only in the non-static method body or constructor body.
- ✓ It should not be used in the static context.

❖ Constructor calling

- ✓ A constructor of a class can make a call to other constructor of the same class or super class constructor.
- ✓ Calling statements are used to call the constructor
- ✓ It is of 2 types
- ✓ this() – is used to call the constructor of the current class.
- ✓ super() – is used to call the constructor of super class.
- ✓ Calling statements can be used to call either no argument constructor or parameterized constructor.
- ✓ Calling of **super class constructor** can be either by **implicit** or **explicit**.
- ✓ The **implicit** calling is provided by the compiler only in the condition, if the super class is having **no argument constructor.**
- ✓ If the super class is having parameterised constructor or the sub class wants to make a call to the parameterised constructor of super class, then sub class constructor must make **explicit call**.

❖ Rules to use calling statements

- ✓ Calling statements must be 1st statement in the constructor body.
- ✓ More than one calling statement is not allowed in one constructor body.
- ✓ That is, it allow to use either this() or super() with in one constructor body.

❖ Constructor chaining

- ✓ Constructor chaining is a phenomenon of constructor calling another constructor, the called constructor calls other constructors.
- ✓ The calling can be with in the same class or from the sub class to super class.
- ✓ Whenever we inherits the class, constructor chaining phenomenon should happens either implicitly or explicitly.

❖ Method Overloading

- ✓ If Methods names are **same** with different arguments then it is called as "Method Overloading".
- ✓ Arguments **must** be differ either in the form of number of argument or in the type.
- ✓ Overloading can happen in both "**Same Class**" as well as "**Sub Class**"

❖ Method Overriding

- ✓ Method Overriding is the process of **hiding the super class method implementation in sub class**.
- ✓ It happens **only** in **Sub Class.**
- ✓ Hence Sub-Class should have **same method signature[method name and arguments]** as Super-Class and the following changes are allowed
  - o We can increase the **visibility** of the methods
  - o return type **should** be same for **Primitive types** and it can be of **covariant type** for **Non-Primitive types**
  - o We can decrease the visibility of the **Exceptions.**
- ✓ If we made changes to arguments, then it becomes **Method Overloading**.

❖ Type Casting

- ✓ Converting one type of information to other type
- ✓ We can perform two types of casting in java:
  - Data Type Casting
  - Class Type Casting

➢ Data Type Casting

- ✓ A Data type is converted to another data type.
- ✓ It can be performed in two ways:
  - Widening
  - Narrowing
- ✓ Converting a lower type to any of the higher type is known as **Widening.**
- ✓ Converting a higher type to any of the lower type is known as **Narrowing.**

➢ Widening

- ✓ Widening can be performed either implicitly or explicitly.
- ✓ If compiler performs widening on its own then it is called as "**Implicit Widening".**
- ✓ If programmer performs widening operation then it is called as "**Explicit Widening**".
- ✓ Ex:

```
int x = 25;                    int x = 25;
double y = 39.49;              double y = 39.49;
double z = x;                  double z =(double)x;
```

➢ Narrowing

- ✓ While performing the **narrowing** operation we lose the precision data.
- ✓ Hence narrowing should be performed explicitly.
- ✓ Ex:

```
int x = 20;
double y = 59.35;
int z = (int)y;
```