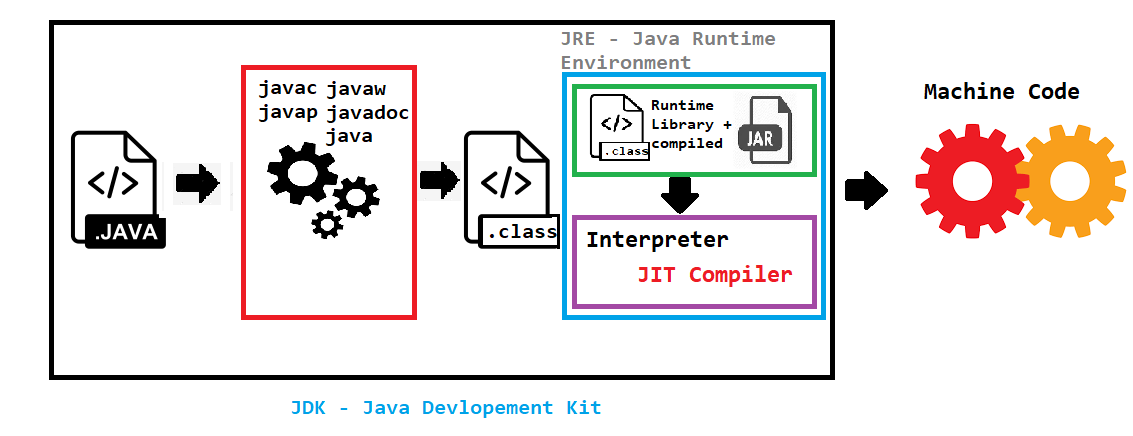
1. **JDK vs JRE**

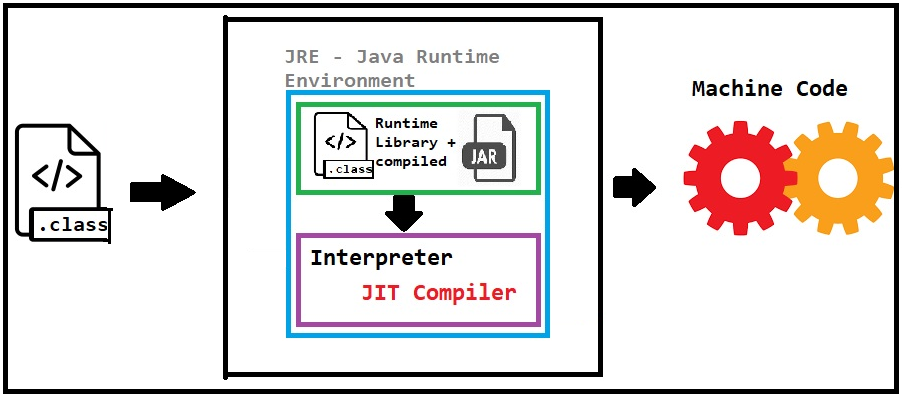
**JDK (Java Development Kit):-**

* Java Development Kit provides all the tools, executables, and binaries required to compile, debug, and execute a Java Program.



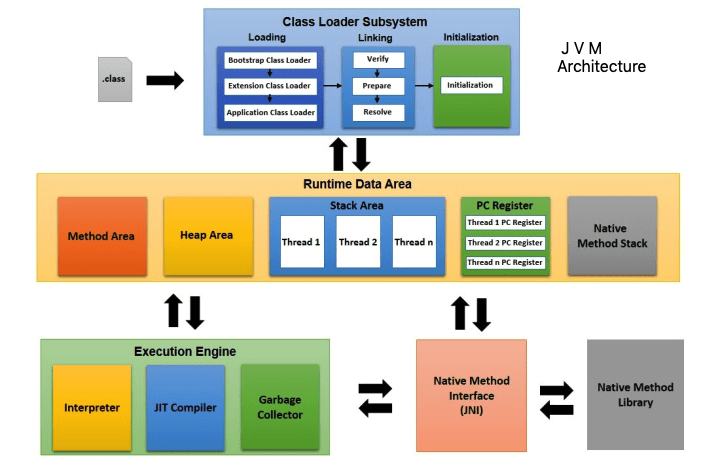
**JRE (Java Runtime Environment)**

* JRE consists of JVM, Java binaries, and other classes to execute java any program



**JVM:**

* JVM is responsible for converting the byte code to the machine-specific code.



Method Area:-

All the static members – static variable, static method, static block

Heap Area:-

Instance members -> instance variable, instance method

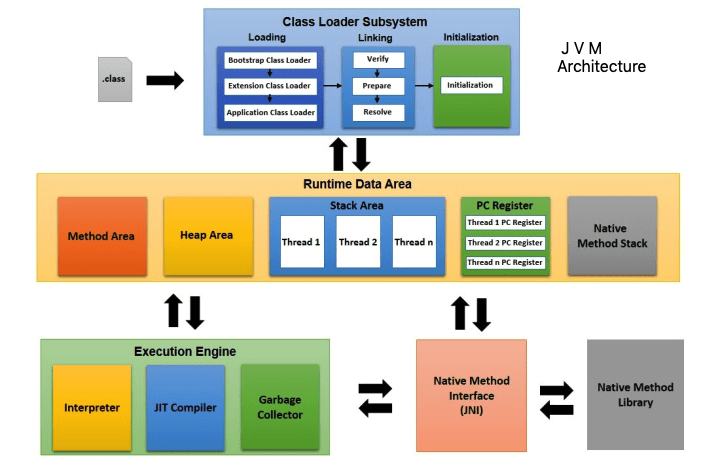
Create anything with new keyword

Stack Area:-

Local variable, object reference

**Just-in-time Compiler (JIT)**

* JIT is part of the JVM that optimizes the process of converting byte code to machine-specific language
* reduces the overall time taken for the compilation of byte code to machine-specific language



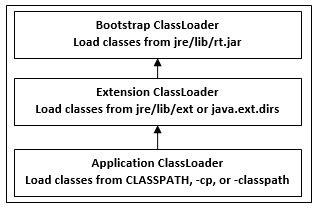
1. Bootstrap class loader

* It loads standard JDK class files from rt.jar (java base) and other core classes.
* It is a parent of all class loaders
* It loads class files from jre/lib/rt.jar(java.base)

1. Extension class loader

* it loads classes from jre/lib/ext directory or any other directory as java.ext.dirs

1. Application class loader
   * It loads application specific classes from the CLASSPATH environment variable
   * It can be set while invoking program using -cp or classpath command line options



**Garbage Collector:**

Automatic Garbage Collector

System.gc();

Finalize(){

Release the resources

}

Java class Scanner -> read the input from the user -> keyboard

Java class FileReader -> read the data from the file -> Hard disk

Gc -> close

s.Close()/ fr.close/ dbconn.close()

Object Oriented Programming Concepts:-

https://classroom.udacity.com/courses/ud123/lessons/1b369991-f1ca-4d6a-ba8f-e8318d76322f/concepts/68d60539-7530-4c25-b038-41c81bc677c2#

final ->

, finally,

finalizer -> before any object being sweep off from the memory finalize method executes

class Test{

protected void finalize(){

}

}

Test t1=new Test();

t1=null 🡪 GC (finalize )

Assembly level language -> add R1, R2

C -> functional or structural or procedure

Why OOPS:-

Classes and objects -> Real time scenario

Reusability

Maintainability

Security

Pillars OOPS:-

Data Hiding – hide the data -> access modifiers -> facebook , bank account

Abstraction – hiding the logic or implementation -> abstract class or interface -> Car, Engine

Encapsulation -> wrapping of related data and method together -> class

Class Employee{

Private Variables

Public getters / setters

}

Polymorphism

Poly – many forms

Giftbox

sendGiftBox() – Task

Method overloading – same method name diff arg

sendGiftBox(indiapost){

goto branch

make a ceil

write an address

payment

return back

}

sendGiftBox(courier){

ask courier service to collect my box

deliver to the customer

}

sendGiftBox(bymyfriend){

}

sendGiftBox(bymyself){

}

Method overriding – same method name and same no of arg

Class Send{

sendGiftBox(bymyfriend){

goto the friend location

wrap the gift

ask the friend deliver to the customer

Bus

}

}

Class SendCourier extends Send{

sendGiftBox(bymyfriend){

ask the friend come down to your loc

wrap the gift by frient

ask the friend deliver to the customer

Car

}

}

Two types of polymorphism:-

Method binding:-

For any method invocation or call, what method definition has to be executed?

Compile time or run time

Compile time polymorphism

Method overriding -> redefine the logic of parent class method into the child class method

Inheritance

Casting

* + Type casting -> converting one primitive type to other primitive type
  + Object casting -> assigning memory child class to the parent class reference vice versa

Overloadding:-

Instance method - Yes

Static method – Yes

Final method – Yes

Overridding:-

Instance method – Yes

== , equals -> hashcode, equals

**Cart -> Object**

**ProductId - 100**

**ProductCost - 30000**

**NoOfProducts -3**

**Serialization -> state of an object 🡪 file system, database**

**Store -> file system, db,**

**Hiberante, JPA**

**Serialization in Java** is a mechanism of writing the state of an object into a byte-stream. It is mainly used in Hibernate, RMI, JPA, EJB and JMS technologies.

The reverse operation of serialization is called deserialization where byte-stream is converted into an object. The serialization and deserialization process is platform-independent, it means you can serialize an object on one platform and deserialize it on a different platform.

For serializing the object, we call the **writeObject()** method of ObjectOutputStream class, and for deserialization we call the **readObject()** method of ObjectInputStream class.

## java.io.Serializable interface

**Serializable** is a marker interface (has no data member and method). It is used to "mark" Java classes so that the objects of these classes may get a certain capability. The **Cloneable** and **Remote** are also marker interfaces.

The **Serializable** interface must be implemented by the class whose object needs to be persisted.

The String class and all the wrapper classes implement the java.io.Serializable interface by default.

**Student.java**

1. **import** java.io.Serializable;
2. **public** **class** Student **implements** Serializable{
3. **int** id;
4. String name;
5. **public** Student(**int** id, String name) {
6. **this**.id = id;
7. **this**.name = name;
8. }
9. }

## Example of Java Serialization

In this example, we are going to serialize the object of **Student** class from above code. The writeObject() method of ObjectOutputStream class provides the functionality to serialize the object. We are saving the state of the object in the file named f.txt.

**Persist.java**

1. **import** java.io.\*;
2. **class** Persist{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. //Creating the object
6. Student s1 =**new** Student(211,"ravi");
7. //Creating stream and writing the object
8. FileOutputStream fout=**new** FileOutputStream("f.txt");
9. ObjectOutputStream out=**new** ObjectOutputStream(fout);
10. out.writeObject(s1);
11. out.flush();
12. //closing the stream
13. out.close();
14. System.out.println("success");
15. }**catch**(Exception e){System.out.println(e);}
16. }
17. }

### **Example of Java Deserialization**

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization. Let's see an example where we are reading the data from a deserialized object.

Deserialization is the process of reconstructing the object from the serialized state. It is the reverse operation of serialization. Let's see an example where we are reading the data from a deserialized object.

**Depersist.java**

1. **import** java.io.\*;
2. **class** Depersist{
3. **public** **static** **void** main(String args[]){
4. **try**{
5. //Creating stream to read the object
6. ObjectInputStream in=**new** ObjectInputStream(**new** FileInputStream("f.txt"));
7. Student s=(Student)in.readObject();
8. //printing the data of the serialized object
9. System.out.println(s.id+" "+s.name);
10. //closing the stream
11. in.close();
12. }**catch**(Exception e){System.out.println(e);}
13. }
14. }