Java Programming Language

Question Bank

Short Questions

1. **Define monitor.**

Ans: a **monitor** is a facility which **monitors** the threads' access to the special room. It ensures that only one thread can access the protected data or code.

2. **How can we create an object of interface?**

Ans: we can't **create an object** of an **Interface** ,we use an **Interface** to hide the implementations from user.

we can **make** reference of it that refers to the **Object** of its implementing class.

3. **What is JVM?**

Ans: JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

4. **Java is robust- Justify.**

Ans: **Java is robust** because it utilizes strong memory management. There is an absence of pointers that bypasses security dilemmas. There is automatic garbage collection in **Java** which runs on the **Java** Virtual Machine to eliminate objects which are not being accepted by a **Java** application anymore.

5. **Differentiate paint() and repaint() method.**

Ans: **paint()** get called automatically at runtime. If you want to call **paint()** manually(again) then **repaint()** is used. The **paint() method** contains instructions for **painting** the specific component. The **repaint() method**, which can't be overridden, is more specific: it controls the update**()** to **paint()** process.

6. **Write significance of CLASSPATH.**

Ans: **Classpath** is a parameter in the **Java** Virtual Machine or the **Java** compiler that specifies the location of user-defined classes and packages. The parameter may be set either on the command-line, or through an environment variable.

7. **List any 4 built in package with brief description.**

Ans: .Some of the commonly used built-in packages are:

1) **java.lang:** Contains language support classes(e.g classed which defines primitive data types, math operations). This package is automatically imported.
2) **java.io:** Contains classed for supporting input / output operations.
3) **java.util:** Contains utility classes which implement data structures like Linked List, Dictionary and support ; for Date / Time operations.
4) **java.applet:** Contains classes for creating Applets.
5) **java.awt:** Contain classes for implementing the components for graphical user interfaces (like button , ;menus etc).

8. **Explain significance of 'public' and 'static' keywords in main() method.**

Ans: It is made **public** so that JVM can invoke it from outside the class as it is not present in the current class. **Static**: It is a **keyword** which is when associated with a **method**, makes it a class related **method**. The **main() method** is **static** so that JVM can invoke it without instantiating the class.

9. Define BYTECODE.
Ans: **Java bytecode** is the instruction set for the **Java** Virtual Machine. It acts similar to an assembler which is an alias representation of a C++ code. As soon as a **java** program is compiled, **java bytecode** is generated.

10. Differentiate finalize() and destructor
Ans: The main difference between finalize() method and destructor() in c++ is that the destructor() is always called when object goes out of scope but in java finalize() method called by garbage collector sporadically before garbage collector free the objects when no reference to that objects exists.

11. Write down use of this pointer.
Ans: this is a keyword in **Java**. It can be used inside the method or constructor of a class. It(this) works as a reference to the current object, whose method or constructor is being invoked. This keyword can be used to refer to any member of the current object from within an instance method or a constructor.

12. Differentiate break and label break.
Ans: There are only one **difference between** the **break and label**. When we used only **break** statement it will **break** the inner loop only according to the condition is given but when we applies **label** with **break** it will **break** the outer loop according to the condition is given.

13. Explain isAlive() and join() method.
Ans: The **isAlive()** method of thread class tests if the thread is alive. A thread is considered alive when the start() method of thread class has been called and the thread is not yet dead. This method returns true if the thread is still running and not finished.
Syntax : public final boolean isAlive()

The **join()** method of thread class waits for a thread to die. It is used when you want one thread to wait for completion of another. This process is like a relay race where the second runner waits until the first runner comes and hand over the flag to him.
Syntax : public final void join()throws InterruptedException

14. Differentiate init() and start() methods of Applet.
Ans: **public void init():** is used to initialized the Applet. It is invoked only once.

public void start(): is invoked after the init() method or browser is maximized. It is used to start the Applet. It is invoked more than once.

15. Explain checked and unchecked Exception.
Ans: **1) Checked:** are the exceptions that are checked at compile time. If some code within a method throws a checked exception, then the method must either handle the exception or it must specify the exception using *throws* keyword.

2) Unchecked are the exceptions that are not checked at compiled time. In Java exceptions under *Error* and *RuntimeException* classes are unchecked exceptions, everything else under throwable is checked.

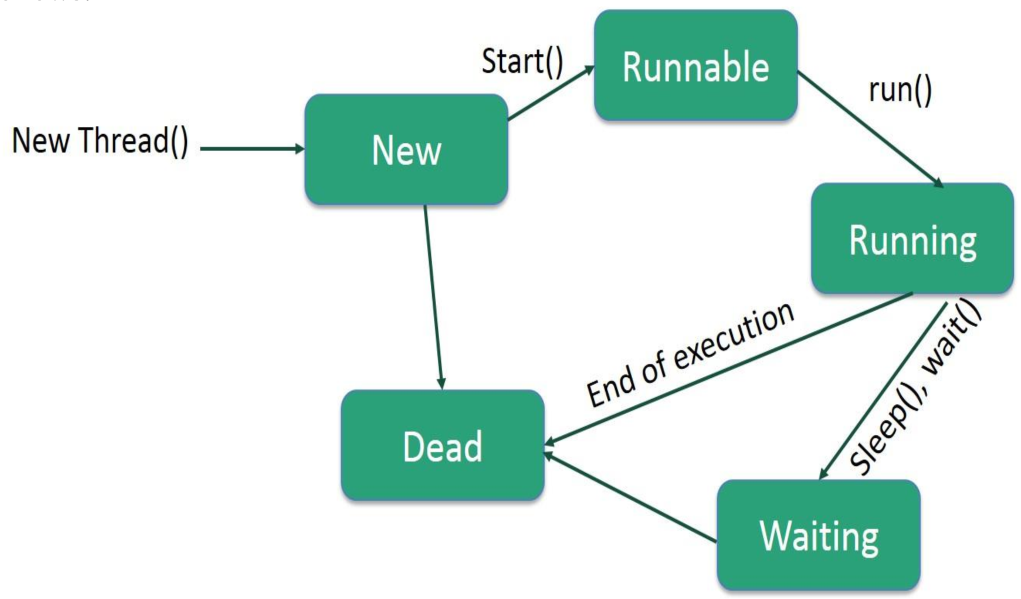
16. How do we compare two objects in java?
Ans: To be able to **compare two Java objects** of the same class the boolean equals(**Object** obj) method must be overriden and implemented by the class. The implementor decides which values must be equal to consider **two objects** to be equal.

17. How access modifier 'default' is different than 'public' modifier?
Ans: **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
Public: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Long Questions

1. Thread Lifecycle.

The life cycle of the thread in java is controlled by JVM. The java thread states are as follows:



- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

2. What is Interface? How it differs from abstract class?

An interface is a reference type in Java. It is similar to class. It is a collection of abstract methods. A class implements an interface, thereby inheriting the abstract methods of the interface.

Along with abstract methods, an interface may also contain constants, default methods, static methods, and nested types. Method bodies exist only for default methods and static methods.

Writing an interface is similar to writing a class. But a class describes the attributes and behaviors of an object. And an interface contains behaviors that a class implements.

Unless the class that implements the interface is abstract, all the methods of the interface need to be defined in the class.

An interface is different from a class in several ways, including −

- You cannot instantiate an interface.
- An interface does not contain any constructors.
- All of the methods in an interface are abstract.
- An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
- An interface is not extended by a class; it is implemented by a class.
- An interface can extend multiple interfaces.

3. Explain various methods of StringBuffer class.

Java 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.

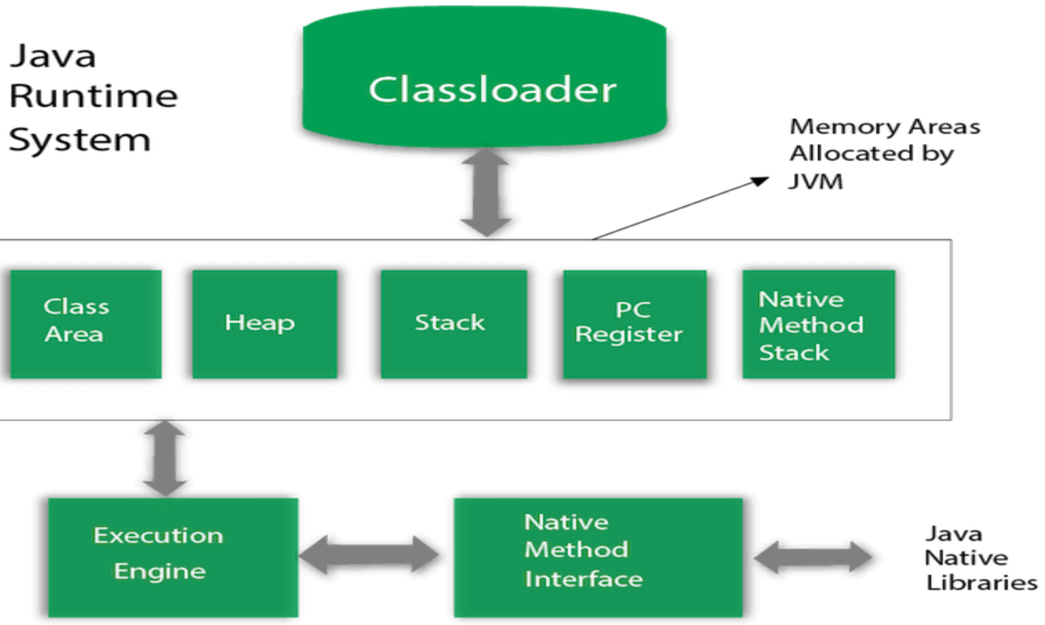
Important methods of StringBuffer class

Modifier and Type	Method	Description
public synchronized StringBuffer	append(String s)	is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
public synchronized StringBuffer	insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
public synchronized StringBuffer	replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
public synchronized StringBuffer	delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.
public synchronized StringBuffer	reverse()	is used to reverse the string.
public int	capacity()	is used to return the current capacity.
public void	ensureCapacity(int minimumCapacity)	is used to ensure the capacity at least equal to the given minimum.
public char	charAt(int index)	is used to return the character at the specified position.
public int	length()	is used to return the length of the string i.e. total number of characters.
public String	substring(int beginIndex)	is used to return the substring from the specified beginIndex.
public String	substring(int	is used to return the substring from the specified

	beginIndex, int endIndex)	beginIndex and endIndex.

4. Explain structure of JVM./Explain java architecture.
JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).



1) Classloader

Classloader is a subsystem of JVM which is used to load class files. Whenever we run the java program, it is loaded first by the classloader. There are three built-in classloaders in Java.

1. **Bootstrap ClassLoader**: This is the first classloader which is the super class of Extension classloader. It loads the *rt.jar* file which contains all class files of Java Standard Edition like java.lang package classes, java.net package classes, java.util package classes, java.io package classes, java.sql package classes etc.
2. **Extension ClassLoader**: This is the child classloader of Bootstrap and parent classloader of System classloader. It loades the jar files located inside *$JAVA_HOME/jre/lib/ext* directory.
3. **System/Application ClassLoader**: This is the child classloader of Extension classloader. It loads the classfiles from classpath. By default, classpath is set to current directory. You can change the classpath using "-cp" or "-classpath" switch. It is also known as Application classloader.

2) Class(Method) Area
Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

3) Heap

It is the runtime data area in which objects are allocated.

4) Stack
Java Stack stores frames. It holds local variables and partial results, and plays a part in method invocation and return.

Each thread has a private JVM stack, created at the same time as thread.

A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes.

5) Program Counter Register
PC (program counter) register contains the address of the Java virtual machine instruction currently being executed.

6) Native Method Stack
It contains all the native methods used in the application.

7) Execution Engine
It contains:

1. **A virtual processor**
2. **Interpreter:** Read bytecode stream then execute the instructions.
3. **Just-In-Time(JIT) compiler:** It is used to improve the performance. JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation. Here, the term "compiler" refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU.

8) Java Native Interface
Java Native Interface (JNI) is a framework which provides an interface to communicate with another application written in another language like C, C++, Assembly etc. Java uses JNI framework to send output to the Console or interact with OS libraries.

5. **Explain method overloading and method overriding in detail./Runtime polymorphism**

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**.

If we have to perform only one operation, having same name of the methods increases the readability of the program.

Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

Advantage of method overloading

Method overloading *increases the readability of the program*.

Different ways to overload the method

There are two ways to overload the method in java

1. By changing number of arguments
2. By changing the data type

If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in Java**.

In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

Usage of Java Method Overriding

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Rules for Java Method Overriding

1. The method must have the same name as in the parent class
2. The method must have the same parameter as in the parent class.
3. There must be an IS-A relationship (inheritance).

6. **What is applet? Explain Graphics methods of Applet.**

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.

9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example:

```
import java.applet.Applet;
import java.awt.*;

public class GraphicsDemo extends Applet{

public void paint(Graphics g){
g.setColor(Color.red);
g.drawString("Welcome",50, 50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);

g.setColor(Color.pink);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);

}
}
/* <html>
<body>
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
</body>
</html>
*/
```

7. What is exception? Explain user defined exception with example.

The **Exception Handling in Java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

If you are creating your own Exception that is known as custom exception or user-defined exception. Java custom exceptions are used to customize the exception according to user need.

By the help of custom exception, you can have your own exception and message.

example of java custom exception

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 }
}

class TestCustomException1{
```

```
static void validate(int age)throws InvalidAgeException{
  if(age<18)
   throw new InvalidAgeException("not valid");
  else
   System.out.println("welcome to vote");
}

public static void main(String args[]){
   try{
   validate(13);
   }catch(Exception m){System.out.println("Exception occured: "+m);}

   System.out.println("rest of the code...");
 }
}
```

```
Output:Exception occured: InvalidAgeException:not valid
       rest of the code...
```

8. Access specifiers with package.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

There are many non-access modifiers, such as static, abstract, synchronized, native, volatile, transient, etc. Here, we are going to learn the access modifiers only.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

9. Explain static method, static variable and block in detail with example.

The **static keyword** in Java is used for memory management mainly. We can apply static keyword with variables, methods, blocks and nested classes. The static keyword belongs to the class than an instance of the class.

The static can be:

1. Variable (also known as a class variable)
2. Method (also known as a class method)
3. Block

1) Java static variable

If you declare any variable as static, it is known as a static variable.

- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

Example:

```
class Student{
  int rollno;//instance variable
  String name;
  static String college ="ITS";//static variable
  //constructor
  Student(int r, String n){
  rollno = r;
  name = n;
  }
  //method to display the values
  void display (){System.out.println(rollno+" "+name+" "+college);}
}
//Test class to show the values of objects
public class TestStaticVariable1{
 public static void main(String args[]){
 Student s1 = new Student(111,"Karan");
 Student s2 = new Student(222,"Aryan");
 //we can change the college of all objects by the single line of code
 //Student.college="BBDIT";
 s1.display();
 s2.display();
 }
}
```

Java static method

If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.

```
//Java Program to demonstrate the use of a static method.
class Student{
    int rollno;
    String name;
    static String college = "ITS";
    //static method to change the value of static variable
```

```
        static void change(){
        college = "BBDIT";
        }
        //constructor to initialize the variable
        Student(int r, String n){
        rollno = r;
        name = n;
        }
        //method to display values
        void display(){System.out.println(rollno+" "+name+" "+college);}
    }

    //Test class to create and display the values of object
    public class TestStaticMethod{
       public static void main(String args[]){
       Student.change();//calling change method
       //creating objects
       Student s1 = new Student(111,"Karan");
       Student s2 = new Student(222,"Aryan");
       Student s3 = new Student(333,"Sonoo");
       //calling display method
       s1.display();
       s2.display();
       s3.display();
       }
    }
```

Java static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of classloading.

Example of static block

```
class A2{
  static{System.out.println("static block is invoked");}
  public static void main(String args[]){
   System.out.println("Hello main");
  }
}
```

10. Explain bitwise operators in detail.

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs the bit-by-bit operation.

The following table lists the bitwise operators −

Assume integer variable A holds 60 and variable B holds 13 then −

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101

Operator	Description	Example
^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

11. Explain array of objects in detail with suitable example.

Arrays are capable of storing objects also. For example, we can create an array of Strings which is a reference type variable.

For example, we will use a class Student containing a single instance variable marks. Following is the definition of this class.

```
class Student {
  int marks;
}
```

An array of objects is created just like an array of primitive type data items in the following way.

```
Student[] studentArray = new Student[7];
```

The above statement creates the array which can hold references to seven Student objects. It doesn't create the Student objects themselves. They have to be created separately using the constructor of the Student class. The studentArray contains seven memory spaces in which the address of seven Student objects may be stored.
The Student objects have to be instantiated using the constructor of the Student class and their references should be assigned to the array elements in the following way.

```
studentArray[0] = new Student();
```

In this way, we create the other Student objects also. If each of the Student objects have to be created using a different constructor, we use a statement similar to the above several times.

12. Explain features of JAVA.

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features which play an important role in the popularity of this language.

A list of most important features of Java language is given below.

1. Simple
2. Object-Oriented
3. Portable
4. Platform independent

5. Secured
6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

Simple

Java is very easy to learn, and its syntax is simple, clean and easy to understand. According to Sun, Java language is a simple programming language because:

- 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.

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:

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

Platform Independent

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. 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 a software-based platform.

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:

1. Runtime Environment
2. API(Application Programming Interface)

Secured

Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:

- **No explicit pointer**
- **Java Programs run inside a virtual machine sandbox**
- **Classloader:** Classloader in Java is a part of the Java Runtime Environment(JRE) which is used to load Java classes into the Java Virtual Machine dynamically. It adds security by separating the package for the classes of the local file system from those that are imported from network sources.
- **Bytecode Verifier:** It checks the code fragments for illegal code that can violate access right to objects.
- **Security Manager:** It determines what resources a class can access such as reading and writing to the local disk.

Robust

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.

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.

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.

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.

13. List and Explain various java tokens

Tokens in Java are the small units of code which a Java compiler uses for constructing those statements and expressions. Java supports 5 types of tokens which are:

1. Keywords
2. Identifiers
3. Literals
4. Operators
5. Special Symbols

Keywords

Keywords in Java are predefined or reserved words that have special meaning to the Java compiler. Each keyword is assigned a special task or function and cannot be changed by the user. You cannot use keywords as variables or identifiers as they are a part of Java syntax itself. A keyword should always be written in lowercase as Java is a case sensitive language. Java supports various keywords, some of them are listed below:

01. abstract	02. boolean	03. byte	04. break	05. class
06. case	07. catch	08. char	09. continue	10. default
11. do	12. double	13. else	14. extends	15. final
16. finally	17. float	18. for	19. if	20. implements
21. import	22. instanceof	23. int	24. interface	25. long
26. native	27. new	28. package	29. private	30. protected
31. public	32. return	33. short	34. static	35. super
36. switch	37. synchronized	38. this	39. throw	40. throws
41. transient	42. try	43. void	44. volatile	45. while
46. assert	47. const	48. enum	49. goto	50. strictfp

Identifier

Java Identifiers are the user-defined names of variables, methods, classes, arrays, packages, and interfaces. Once you assign an identifier in the Java program, you can use it to refer the value associated with that identifier in later statements. There are some de facto standards which you must follow while naming the identifiers such as:

- Identifiers must begin with a letter, dollar sign or underscore.
- Apart from the first character, an identifier can have any combination of characters.
- Identifiers in Java are case sensitive.
- Java Identifiers can be of any length.
- Identifier name cannot contain white spaces.
- Any identifier name must not begin with a digit but can contain digits within.
- Most importantly, **keywords** can't be used as identifiers in Java.

Example:

```
//Valid Identifiers
$myvariable  //correct
_variable    //correct
variable     //correct
edu_identifier_name //correct
```

```
edu2019var   //correct

//Invalid Identifiers
edu variable    //error
Edu_identifier  //error
&variable       //error
23identifier    //error
switch          //error
var/edu         //error
edureka's       //error
```

Literals

Literals in Java are similar to normal variables but their values cannot be changed once assigned. In other words, literals are constant variables with fixed values. These are defined by users and can belong to any data type. Java supports five types of literals which are as follows:

1. Integer
2. Floating Point
3. Character
4. String
5. Boolean

Operators

An operator in Java is a special symbol that signifies the compiler to perform some specific mathematical or non-mathematical operations on one or more operands. Java supports 8 types of operators. Below I have listed down all the operators, along with their examples:

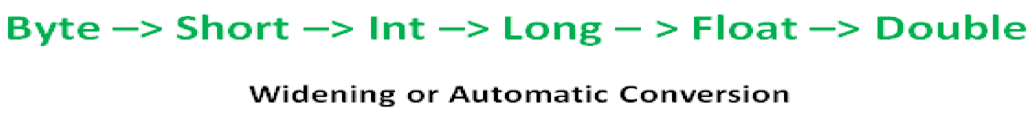
Operator	Examples
Arithmetic	**+ , – , / , * , %**
Unary	**++ , – – , !**
Assignment	**= , += , -= , *= , /= , %= , ^=**
Relational	**==, != , < , >, <= , >=**
Logical	**&& , ||**
Ternary	**(Condition) ? (Statement1) : (Statement2);**
Bitwise	**& , | , ^ , ~**
Shift	**<< , >> , >>>**

14. Explain type casting with suitable example.

When you assign value of one data type to another, the two types might not be compatible with each other. If the data types are compatible, then Java will perform the conversion automatically known as Automatic Type Conversion and if not then they need to be casted or converted explicitly. For example, assigning an int value to a long variable.

Widening conversion takes place when two data types are automatically converted. This happens when:

- The two data types are compatible.
- When we assign value of a smaller data type to a bigger data type.


Byte –> Short –> Int –> Long – > Float –> Double

Widening or Automatic Conversion

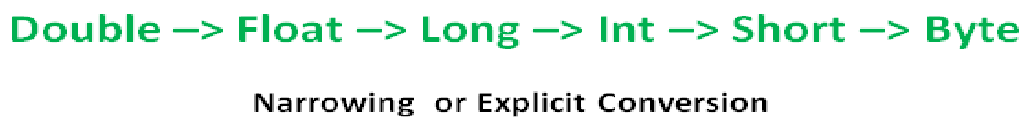
Example:

```
class Test
{
   public static void main(String[] args)
   {
      int i = 100;

      // automatic type conversion
      long l = i;

      // automatic type conversion
      float f = l;
      System.out.println("Int value "+i);
      System.out.println("Long value "+l);
      System.out.println("Float value "+f);
   }
}
```

Narrowing or Explicit Conversion

If we want to assign a value of larger data type to a smaller data type we perform explicit type casting or narrowing.

- This is useful for incompatible data types where automatic conversion cannot be done.
- Here, target-type specifies the desired type to convert the specified value to.


Double –> Float –> Long –> Int –> Short –> Byte
Narrowing or Explicit Conversion

```
//Java program to illustrate explicit type conversion
class Test
{
   public static void main(String[] args)
   {
      double d = 100.04;

      //explicit type casting
      long l = (long)d;

      //explicit type casting
      int i = (int)l;
      System.out.println("Double value "+d);

      //fractional part lost
      System.out.println("Long value "+l);

      //fractional part lost
      System.out.println("Int value "+i);
   }
}
```

Explicit type casting in Expressions

While evaluating expressions, the result is automatically updated to larger data type of the operand. But if we store that result in any smaller data type it generates compile time error, due to which we

need to type cast the result.
Example:

```
//Java program to illustrate type casting int to byte
class Test
{
   public static void main(String args[])
   {
      byte b = 50;

      //type casting int to byte
      b = (byte)(b * 2);
      System.out.println(b);
   }
}
```

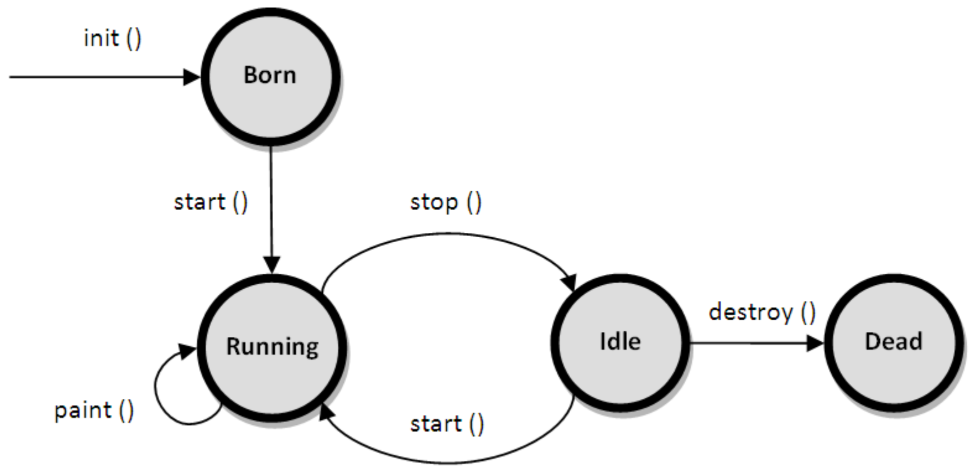
15. Applet Lifecycle.

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

Life Cycle of an Applet

Four methods in the Applet class gives you the framework on which you build any serious applet –

- **init** − This method is intended for whatever initialization is needed for your applet. It is called after the param tags inside the applet tag have been processed.
- **start** − This method is automatically called after the browser calls the init method. It is also called whenever the user returns to the page containing the applet after having gone off to other pages.
- **stop** − This method is automatically called when the user moves off the page on which the applet sits. It can, therefore, be called repeatedly in the same applet.
- **destroy** − This method is only called when the browser shuts down normally. Because applets are meant to live on an HTML page, you should not normally leave resources behind after a user leaves the page that contains the applet.
- **paint** − Invoked immediately after the start() method, and also any time the applet needs to repaint itself in the browser. The paint() method is actually inherited from the java.awt.



16. Inter Thread communication and thread synchronization

Synchronization in java is the capability *to control the access of multiple threads to any shared resource*.

Java Synchronization is better option where we want to allow only one thread to access the shared resource.

There are two types of thread synchronization mutual exclusive and inter-thread communication.

1. Mutual Exclusive
    1. Synchronized method.
    2. Synchronized block.
    3. static synchronization.
2. Cooperation (Inter-thread communication in java)

Inter-thread communication or **Co-operation** is all about allowing synchronized threads to communicate with each other.

Cooperation (Inter-thread communication) is a mechanism in which a thread is paused running in its critical section and another thread is allowed to enter (or lock) in the same critical section to be executed.It is implemented by following methods of **Object class**:

- wait()
- notify()
- notifyAll()

1) wait() method
Causes current thread to release the lock and wait until either another thread invokes the notify() method or the notifyAll() method for this object, or a specified amount of time has elapsed.

The current thread must own this object's monitor, so it must be called from the synchronized method only otherwise it will throw exception.

Syntax: public final void wait()throws InterruptedException

2) notify() method
Wakes up a single thread that is waiting on this object's monitor. If any threads are waiting on this object, one of them is chosen to be awakened. The choice is arbitrary and occurs at the discretion of the implementation. Syntax:

public final void notify()

3) notifyAll() method
Wakes up all threads that are waiting on this object's monitor. Syntax:

public final void notifyAll()

Let's see the simple example of inter thread communication.

```
class Customer{
int amount=10000;
synchronized void withdraw(int amount){
System.out.println("going to withdraw...");

if(this.amount<amount){
System.out.println("Less balance; waiting for deposit...");
try{wait();}catch(Exception e){}
}
this.amount-=amount;
System.out.println("withdraw completed...");
}

synchronized void deposit(int amount){
System.out.println("going to deposit...");
this.amount+=amount;
System.out.println("deposit completed... ");
notify();
}
}

class Test{
public static void main(String args[]){
final Customer c=new Customer();
new Thread(){
public void run(){c.withdraw(15000);}
}.start();
new Thread(){
public void run(){c.deposit(10000);}
}.start();

}}
```

17. Final Keyword

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

1. variable
2. method
3. class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example:

There is a final variable speedlimit, we are going to change the value of this variable, but It can't be changed because final variable once assigned a value can never be changed.

```
class Bike9{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[]){
 Bike9 obj=new  Bike9();
 obj.run();
 }
}
Output:Compile Time Error
```

Java final method

If you make any method as final, you cannot override it.

Example:

```
class Bike{
  final void run(){System.out.println("running");}
}

class Honda extends Bike{
   void run(){System.out.println("running safely with 100kmph");}

   public static void main(String args[]){
   Honda honda= new Honda();
   honda.run();
   }
}
      Output:Compile Time Error
```

Java final class

If you make any class as final, you cannot extend it.

Example:

```
final class Bike{}

class Honda1 extends Bike{
  void run(){System.out.println("running safely with 100kmph");}

  public static void main(String args[]){
  Honda1 honda= new Honda1();
  honda.run();
  }
}
Output:Compile Time Error
```

18. Chained Exception

Chained exception helps to relate one exception to other. Often we need to throw a custom exception and want to keep the details of an original exception that in such scenarios we can use the chained exception mechanism. Consider the following example, where we are throwing a custom exception while keeping the message of the original exception.

Example

```
public class Tester {
  public static void main(String[] args) {
    try {
      test();
    }catch(ApplicationException e) {
      System.out.println(e.getMessage());
    }
  }

  public static void test() throws ApplicationException {
    try {
      int a = 0;
      int b = 1;
      System.out.println(b/a);
    }catch(Exception e) {
      throw new ApplicationException(e);
    }
  }
}

class ApplicationException extends Exception {
  public ApplicationException(Exception e) {
    super(e);
  }
}
```

Output

```
java.lang.ArithmeticException: / by zero
```

The throwable class supports chained exception using the following methods:

Constructors

1. **Throwable(Throwable cause)** - the cause is the current exception.
2. **Throwable(String msg, Throwable cause)** - msg is the exception message, the cause is the current exception.

Methods

1. **getCause** - returns actual cause.
2. **initCause(Throwable cause)** - sets the cause for calling an exception.

19. Super Keyword.

The **super** keyword in Java is a reference variable which is used to refer immediate parent class object.

Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

1. super can be used to refer immediate parent class instance variable.
2. super can be used to invoke immediate parent class method.
3. super() can be used to invoke immediate parent class constructor.

1) super is used to refer immediate parent class instance variable.

We can use super keyword to access the data member or field of parent class. It is used if parent class and child class have same fields.

2) super can be used to invoke parent class method

The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

3) super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor.

Example:

Let's see the real use of super keyword. Here, Emp class inherits Person class so all the properties of Person will be inherited to Emp by default. To initialize all the property, we are using parent class constructor from child class. In such way, we are reusing the parent class constructor.

```
class Person{
int id;
String name;
Person(int id,String name){
this.id=id;
this.name=name;
}
}
class Emp extends Person{
float salary;
Emp(int id,String name,float salary){
super(id,name);//reusing parent constructor
this.salary=salary;
}
void display(){System.out.println(id+" "+name+" "+salary);}
}
class TestSuper5{
public static void main(String[] args){
Emp e1=new Emp(1,"ankit",45000f);
e1.display();
}}
```

Output:

1 ankit 45000

20. Applet Tag

An **applet** is a Java program that runs in a Web browser. An applet can be a fully functional Java application because it has the entire Java API at its disposal.

The **<applet>** tag in HTML was used to *embed Java applets into any HTML document*. The **<applet>** tag was deprecated in HTML 4.01, and it's support has been completely discontinued starting from HTML 5.

Syntax:

```
<applet attribute1 attribute2....>
  <param parameter1>
  <param parameter2>
  ....
</applet>
```

Attributes available to be used in conjunction with the **<applet>** tag are as follows:
The <applet> tag takes a number of attributes, with one of the most important being the **code** attribute. This **code** attribute is used to link a Java applet to the concerned HTML document. It specifies the file name of the Java applet.

ATTRIBUTE NAME	VALUES	REMARKS
align	left right top bottom middle baseline	Specifies the alignment of an applet.
alt	text	Specifies an alternate text for an applet
border	pixels	Specifies the border around the applet panel
codebase	URL	Specifies a relative base URL for applets specified in the code attribute
height	pixels	Specifies the height of an applet
hspace	pixels	Defines the horizontal spacing around an applet
name	name	Defines the name for an applet (to use in scripts)
vspace	pixels	Defines the vertical spacing around an applet

ATTRIBUTE NAME	VALUES	REMARKS
width	pixels	Specifies the width of an applet

21. What is applet? How to pass parameters to the applet? Explain with example.

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plugin is required at client browser to execute applet.

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter(). Syntax:

```
public String getParameter(String parameterName)
```

Example:

```
import java.applet.Applet;
import java.awt.Graphics;

public class UseParam extends Applet{

public void paint(Graphics g){
String str=getParameter("msg");
g.drawString(str,50, 50);
}

}
```

myapplet.html

```
<html>
<body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
```

```
</html>
```