

**1.How to Create an Object in Java?**

=> 1. We can use the "new" operator to create an object.   
 2. There is no "delete" operator in java because destruction of useless objects is the responsibility of the garbage collector.

class Test {

String name;

int a;

public static void main(String[] args) {

Test t = new Test();

}

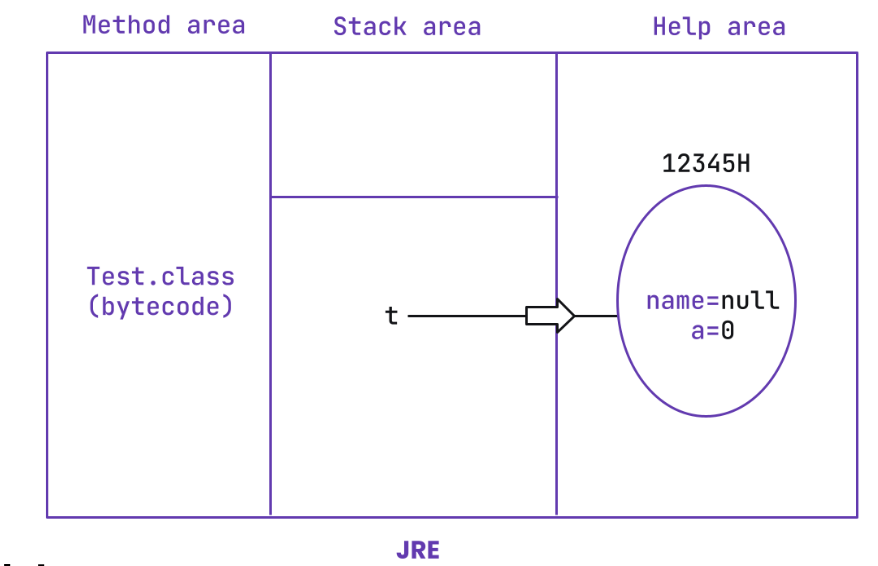
}

**2. What is the use of a new keyword in Java?**

=> new is an operator to create an object , if we know the class name at the beginning then we can create an object by using new operator.

When we say new operator , JVM would allocate memory on the heap area then JVM will load the supplied class name data into the method area and JVM will initialise the memory of instance variables.

JVM will set the default value for instance variables based on the data type. Once the memory is set then the address of the object will be stored inside the reference variable.



**3. What are the different types of variables in Java?**

=> **Based on the type** of value represented by a variable all variables are divided into 2 types.

They are:

**1. Primitive variables**

**2.** **Reference variables**

**Primitive variables:** Primitive variables can be used to represent primitive values.

Example: int x=10;

**Reference variables:** Reference variables can be used to refer objects.

Example: Student s=new Student();

**Based on the behaviour and position** of declaration all variables are divided into the following 3 types.

1. **Instance variables**
2. **Static variables**
3. **Local variables**

**Instance variables:**

* If the value of a variable is varied from object to object such types of variables are called instance variables.
* For every object a separate copy of instance variables will be created.
* Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly the same as scope of objects.
* Instance variables will be stored on the heap as the part of the object.
* Instance variables should be declared within the class directly but outside of any method or block or constructor.
* Instance variables can be accessed directly from the Instance area. But cannot be accessed directly from a static area.
* But by using object reference we can access instance variables from static area .

**Example**

class Test {

int i =10;

public static void main(String[] args) {

System.out.println(i);//CE: non static variable can’t be referenced

Test t = new Test();

System.out.println(t.i);//valid

t.m1();

}

public void m1()

{

System.out.println(i);//valid

} }

**Local variables**

* Sometimes to meet temporary requirements of the programmer we can declare variables inside a method or block or constructors such type of variables are called local variables or automatic variables or temporary variables or stack variables.
* Local variables will be stored inside the stack.
* The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local variables is exactly the same as the scope of the block in which we declared.

**Example**

class Test {

public static void main(String[] args) {

int i =0;

}

for (int j =0;j<=3 ;j++ )

{

i = i+j;

System.out.println(j);//CE

}

* The local variables will be stored on the stack.
* For the local variables JVM won't provide any default values, we should perform initialization explicitly before using that variable.
* It is never recommended to perform initialization for the local variables inside logical blocks because there is no guarantee of executing that block always at runtime.
* It is highly recommended to perform initialization for the local variables at the time of declaration at least with default values.
* The only applicable modifier for local variables is final. If we are using any other modifier we will get a compile time error.

**4.What is the difference between Instance variable and Local variables?**

=> **Based on the behaviour and position** of declaration these two are the one of the types of the variable.

**Instance variables:**

* If the value of a variable is varied from object to object such types of variables are called instance variables.
* For every object a separate copy of instance variables will be created.
* Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly the same as scope of objects.
* Instance variables will be stored on the heap as the part of the object.
* Instance variables should be declared within the class directly but outside of any method or block or constructor.
* Instance variables can be accessed directly from the Instance area. But cannot be accessed directly from a static area.
* But by using object reference we can access instance variables from static area .

**Example**

class Test {

int i =10;

public static void main(String[] args) {

System.out.println(i);//CE: non static variable can’t be referenced

Test t = new Test();

System.out.println(t.i);//valid

t.m1();

}

public void m1()

{

System.out.println(i);//valid

} }

**Local variables**

* Sometimes to meet temporary requirements of the programmer we can declare variables inside a method or block or constructors such type of variables are called local variables or automatic variables or temporary variables or stack variables.
* Local variables will be stored inside the stack.
* The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local variables is exactly the same as the scope of the block in which we declared.

**Example**

class Test {

public static void main(String[] args) {

int i =0;

}

for (int j =0;j<=3 ;j++ )

{

i = i+j;

System.out.println(j);//CE

}

* The local variables will be stored on the stack.
* For the local variables JVM won't provide any default values, we should perform initialization explicitly before using that variable.
* It is never recommended to perform initialization for the local variables inside logical blocks because there is no guarantee of executing that block always at runtime.
* It is highly recommended to perform initialization for the local variables at the time of declaration at least with default values.
* The only applicable modifier for local variables is final. If we are using any other modifier we will get a compile time error.

**5.In which area memory is allocated for instance variable and local variable.**

=> **Stack** is a memory place where the methods and the local variables are stored. (variable references either primitive or object references are also stored in the stack).

**Heap** is a memory place where the objects and its instance variable are stored.

**6. What is method overloading?**

* => Two methods are said to be overloaded if and only if both have the same name but different argument types.
* In the 'C' language we can't take 2 methods with the same name and different types. If there is a change in argument type compulsory we should go for a new method name.

**Example :**

abs() for int datatype

labs() for long datatype

fabs() for float datatype.

* Lack of overloading in "C" increases complexity of the programming.
* But in java we can take multiple methods with the same name and different argument types. abs(int) for int datatype

abs(long) for long datatype

abs(float) for float datatype

* Having the same name and different argument types is called method overloading. All these methods are

considered as overloaded methods.

* Having overloading concept in java reduces complexity of the programming

**Example**

class Test {

public void m1(){

System.out.println("zero arg method");

}

public void m1(int i){

System.out.println("int arg method");

}

public void m1(double d){

System.out.println("double arg method");

}

public static void main(String[] args) {

Test t = new Test();

t.m1();

t.m1(10);

t.m1(10.5);

} }

**Output**   
zero arg method int arg method double arg method

**Conclusion :**  
In overloading, the compiler is responsible to perform method resolution(decision) based on the reference type(but not based on run time object). Hence overloading is also considered as compile time polymorphism(or) static polymorphism (or)early binding.