**Q1) What is a class and an object in java?**

Class 🡪 A template that describe the kind of state and behavior its object support

Objects 🡪 At runtime when the JVM encounters a **new** keyword , it will use an appropriate class to make an object which is instance of that class. That object will have its own state and access to all of the behaviors defined by the class.

**Q2) Name some access modifiers and non-access modifiers (access specifires)?**

Access modifiers 🡪

public private

default protected

Non-Access modifiers 🡪

final transient

static synchronized

abstract native

**Q3) List types of variables in java with their default values?**

Variables in java can be

**Primitive** variables

🡪 These are the **char, boolean, byte, short, int, long, float, double**

**Reference** variables

* The variables referencing an object are called reference variables, they **can be static, local or instance**

**Instance** variables

* Variables declared at class level are instance variables
* When a class is instantiated the instance variables are initialized with the below values:

o char - \u0000 (Hex representation of 0)

o boolean – false o long – 0L

o byte – 0 o foat – 0.0f

o short - 0 o double – 0.0d

o int - 0 o String – null

🡪 Instance variables can be **final**, and **can** have **default**, **public**, **private** and **protected** access modifiers.

**Local** variables

* Variables declared inside a method are known as local variables
* They **do not** get and **default** values and cannot be accessed outside the method they are declared in.
* They **cannot** have access modifiers like **public**, **protected**, **private**, **transient**, **abstract** and **static**. However they **can only be final**.

**Note**: **Instance** variable are created in **HEAP** and **local** variables are created in **STACK**

**Q4) What is method overloading?**

Method Overloading means to have two or more methods with same name in the same class with different arguments. The benefit of method overloading is that it allows you to implement methods that support the same semantic operation but differ by argument number or type.  
Note:

* Overloaded methods **MUST** change the **argument list**
* Overloaded methods CAN change the return type
* Overloaded methods CAN change the access modifier
* Overloaded methods CAN declare new or broader checked exceptions

A method can be overloaded in the same class or in a subclass

Q5) **What is Constructor**?

* A constructor is a special method whose task is to initialize the object of its class.
* It is special because its name is the **same as the class name**.
* They do not have return types, not even **void** and therefore they cannot return values.
* They **cannot be inherited**, though a derived class can call the base class constructor.
* Constructor is invoked whenever an object of its associated class is created.
* Constructors can have all the access modifiers , public, private , protected and default

Q6) **How does the Java default constructor be provided?**

If a class defined by the code does **not** have any constructor, compiler will automatically provide one no-parameter-constructor (default-constructor) for the class in the byte code. The access modifier (public/private/etc.) of the default constructor is the same as the class itself.

Q7) **Can constructor be inherited**?

**No**, constructor cannot be inherited, though a derived class can call the base class constructor.

Q8) **What are the differences between Contructors and Methods?**

|  |  |  |
| --- | --- | --- |
|  | **Constructors** | **Methods** |
| **Purpose** | Create an instance of a class | Group Java statements |
| **Modifiers** | Cannot be *abstract, final, native, static*, or *synchronized* | Can be *abstract, final, native, static*, or *synchronized* |
| **Return Type** | No return type, not even void | void or a valid return type |
| **Name** | Same name as the class (first letter is capitalized by convention) -- usually a noun | Any name except the class. Method names begin with a lowercase letter by convention -- usually the name of an action |
| *this* | Refers to another constructor in the same class. If used, it must be the first line of the constructor | Refers to an instance of the owning class. Cannot be used by static methods. |
| *super* | Calls the constructor of the parent class. If used, must be the first line of the constructor | Calls an overridden method in the parent class |
| **Inheritance** | Constructors are not inherited | Methods are inherited |

Q9) **How are this() and super() used with constructors**?

* Constructors use *this* to refer to another constructor in the same class with a different parameter list.
* Constructors use *super* to invoke the superclass's constructor. If a constructor uses *super*, it must use it in the first line; otherwise, the compiler will complain.

**Q10) What are the differences between Class Methods and Instance Methods?**

|  |  |
| --- | --- |
| **Class Methods** | **Instance Methods** |
| Class methods are methods which are declared as **static**. The method can be called without creating an instance of the class | Instance methods on the other hand require an instance of the class to exist before they can be called, so an instance of a class needs to be created by using the **new** keyword. Instance methods operate on specific instances of classes. |
| Class methods can only operate on class members and not on instance members as class methods are unaware of instance members. | Instance methods of the class can also not be called from within a class method unless they are being called on an instance of that class. |

**Q11) What are Access Specifiers?**

One of the techniques in object-oriented programming is ***encapsulation***. It concerns the hiding of data in a class and making this class available only through methods. Java allows you to control access to classes, methods, and fields via so-called ***access specifiers*.**.

**Q12) What are Access Specifiers available in Java?**

Java offers four access specifiers, listed below in decreasing accessibility:

* **Public**- *public* classes, methods, and fields can be accessed from everywhere.
* **Protected**- *protected* methods and fields can only be accessed within the same class to which the methods and fields belong, within its subclasses, and within classes of the same package.
* **Default(no specifier)-** If you do not set access to specific level, then such a class, method, or field will be accessible from inside the same package to which the class, method, or field belongs, but not from outside this package.
* **Private**- *private* methods and fields can only be accessed within the same class to which the methods and fields belong. *private* methods and fields are not visible within subclasses and are not inherited by subclasses.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Situation | public | protected | default | private |
| Accessible to class   from same package? | yes | yes | yes | no |
| Accessible to class   from different package? | yes | no, *unless it is a subclass* | no | no |

Q13) **What is final modifier**?

The final modifier keyword makes that the programmer cannot change the value anymore. The actual meaning depends on whether it is applied to a class, a variable, or a method.

* *final* **Classes**- A final class cannot have subclasses.
* *final* **Variables**- A final variable cannot be changed once it is initialized.
* *final* **Methods**- A final method cannot be overridden by subclasses.

Q15) **What is static block?**

Static block which exactly executed exactly once when the class is first loaded into JVM. Before going to the main method the static block will execute.

Q16) **What are static variables?**

Variables that have only one copy per class are known as static variables. They are not attached to a particular instance of a class but rather belong to a class as a whole. They are declared by using the static keyword as a modifier.

static type varIdentifier;

where, the name of the variable is varIdentifier and its data type is specified by type.

Note: Static variables that are not explicitly initialized in the code are **automatically initialized** with a default value which same as that of instance variables.

Q17) **What is the difference between static and non-static variables?**

A static variable is associated with the class as a whole rather than with specific instances of a class. Non-static variables take on unique values with each object instance.

Q18) **What are static methods?**

Methods declared with the keyword static as modifier are called static methods or class methods. They are so called because they affect a class as a whole, not a particular instance of the class. Static methods are always invoked without reference to a particular instance of a class.  
Note: The use of a static method suffers from the following restrictions:

* A static method can only call other static methods.
* A static method must only access static data.
* A static method **cannot** reference to the current object using keywords *super* or *this*.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Static Methods | Non-Static Methods | Static Variables | Non-Static Variables |
| Static Method | yes | no | yes | no |
| Non-Static Methods | yes | yes | yes | yes |

**Q19) What are the uses of final method?**

There are two reasons for marking a method as final:

* Disallowing subclasses to change the meaning of the method.
* Increasing efficiency by allowing the compiler to turn calls to the method into inline Java code.

**Difference Between ClassNotFoundException Vs NoClassDefFoundError In Java**

|  |  |
| --- | --- |
| CLASSNOTFOUNDEXCEPTION | NOCLASSDEFFOUNDERROR |
| ClassNotFoundException is a checked Exception it extends java.lang.Exception class | NoClassDefFoundError is an Error it extends java.lang.LinkageError class |
| ClassNotFoundException occurs when the application tries to load a class dynamically which is not present in the classpath. | NoClassDefFoundError occurs when the class is found during the compile time but not at the run time |
| ClassNotFoundException occurs by the explicit loading of the class through Class.forName() or ClassLoader.loadClass() or ClassLoader.findSystemClass() | NoClassDefFoundError occurs as a result of implicit loading of class due to a method call or while accessing a other class variable |