1. Difference between JRE, JVM and JDK

Sl.No	JRE	JVM	JDK
1.	JRE is an acronym for Java Runtime Environment. It is also written as Java RTE.	JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides a runtime environment in which Java bytecode can be executed.	JDK (Java Development Kit) is a software development kit to develop applications in Java. In addition to JRE, JDK also contains number of development tools (compilers, JavaDoc, Java Debugger etc.).
2.	JRE is platform dependent.	JVM is platform independent.	JDK is platform dependent i.e for different platforms different JDK required.
3.	JRE contains class libraries and other supporting files that JVM requires to run the program.	JVM does not include software development tools.	It contains tools for developing, debugging and monitoring java application.
4.	JRE = Java Virtual Machine (JVM) + Libraries to run the application.	JVM = Only Runtime environment for executing the Java byte code.	JDK = Java Runtime Environment (JRE) + Development tools.

2. Difference between Local variable , Instance variable and Static variable

Sl.No	Local variable	Instance variable	Static variable
1.	A variable declared inside the body of the method is called local variable.	A variable declared inside the class but outside the body of the method, is called an instance variable.	A variable that is declared as static is called a static variable.It cannot be local.
2.	Local variable can only be within that method and the other methods in the class aren't even aware that the variable exists.	It is called an instance variable because its value is instance-specific and is not shared among instances.	You can create a single copy of the static variable and share it among all the instances of the class.Memory allocation for static variables happens only once when the class is loaded in the memory
3.	A local variable cannot be defined with "static" keyword.	It is not declared as static.	It is declared as static.

3. Difference between Primitive data types and Non-primitive data types

Sl.No	Primitive data types	Non-primitive data types
1.	The primitive data types include boolean, char, byte, short, int, long, float and double.	The non-primitive data types include Classes, Interfaces, and Arrays.
2.	Primitive data types are the building blocks of data manipulation.	Non-Primitive data types are the building blocks of data integration.

4. Difference between static or class method and instance method

Sl.No	static or class method	instance method
1.	A method that is declared as static is known as the static method.	A method that is not declared as static is known as the instance method.
2.	We don't need to create the objects to call the static methods.	The object is required to call the instance methods.
3.	Non-static (instance) members cannot be accessed in the static context (static method, static block, and static nested class) directly.	Static and non-static variables both can be accessed in instance methods.
4.	For example: public static int cube(int n){ return n*n*n;}	For example: public void msg(){...}.

5. Difference between this and super

Sl.No	this	super
1.	this keyword always points to the current class context.	The super keyword always points to the parent class contexts.
2.	this keyword primarily used to differentiate between local and instance variables when passed in the class constructor.	The super keyword is primarily used for initializing the base class variables within the derived class constructor.
3.	Non-static (instance) members cannot be accessed in the static context (static method, static block, and static nested class) directly.	Static and non-static variables both can be accessed in instance methods.
4.	The super and this must be the first statement inside constructor otherwise the compiler will throw an error.	

6. Difference between public, private, protected, default

Sl.No	public	private	protected	default
1.	The access level of a public modifier is everywhere.	The access level of a private modifier is only within the class.	The access level of a protected modifier is within the package and outside the package through child class.	The access level of a default modifier is only within the package.
2.	It can be accessed from within the class, outside the class, within the package and outside the package.	It cannot be accessed from outside the class.	If you do not make the child class, it cannot be accessed from outside the package.	It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

7. Difference between == and equals

Sl.No	==	equals
1.	== is an operator.	.equals() is a method
2.	We can use == operators for reference comparison (address comparison).	.equals() method used for content comparison.
3.	== checks if both objects point to the same memory location.	.equals() evaluates to the comparison of values in the objects.
4.	There is no overridden of ==	If a class does not override the equals method, then by default, it uses the equals(Object o) method of the closest parent class that has overridden this method

8. Difference String, StringBuffer, StringBuilder

Sl.No	String	StringBuffer	StringBuilder
1.	The String class is immutable.	The StringBuffer class is mutable.	The StringBuffer class is mutable.
2.	String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method.	StringBuffer class doesn't override the equals() method of Object class.	StringBuilder class doesn't override the equals() method of Object class.
3.	String is non synchronized.	StringBuffer is synchronized i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously.	StringBuilder is non-synchronized i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously.
4.	String class is slower while performing concatenation operation.	StringBuffer is less efficient than StringBuilder.	StringBuilder is more efficient than StringBuffer.
5.	String class uses String constant pool.	StringBuffer uses Heap memory and was introduced in Java 1.0	StringBuilder uses Heap memory and was introduced in Java 1.5

9. Difference between Method Overloading and Method overriding

Sl.No	Method Overloading	Method overriding
1.	Method overloading is used to increase the readability of the program.	Method overriding is used to provide the specific implementation of the method that is already provided by its super class.
2.	Method overloading is performed within class.	Method overriding occurs in two classes that have IS-A (inheritance) relationship.
3.	In case of method overloading, parameter must be different.	In case of method overriding, parameter must be same.
4.	Method overloading is the example of compile time polymorphism.	Method overriding is the example of run time polymorphism.
5.	In java, method overloading can't be performed by changing return type of the method only. Return type can be same or different in method overloading. But you must have to change the parameter.	Return type must be same or covariant in method overriding.

10. Difference between Abstract class and Interface

Sl.No	Abstract class	Interface
1.	Abstract class can have abstract and non-abstract methods.	Interface can have only abstract methods. Since Java 8, it can have default and static methods also.
2.	Abstract class doesn't support multiple inheritance.	Interface supports multiple inheritance.
3.	Abstract class can have final, non-final, static and non-static variables.	Interface has only static and final variables.
4.	Abstract class can provide the implementation of interface.	Interface can't provide the implementation of abstract class.
5.	The abstract keyword is used to declare abstract class.	The interface keyword is used to declare interface.
6.	An abstract class can extend another Java class and implement multiple Java interfaces.	An interface can extend another Java interface only.
7.	An abstract class can be extended using keyword "extends".	An interface can be implemented using keyword "implements".
8.	A Java abstract class can have class members like private, protected, etc.	Members of a Java interface are public by default.

11. Difference between Static Interface Method and Default Interface Method

Sl.No	Static Interface Method	Default Interface Method
1.	It is a static method which belongs to the interface only. We can write implementation of this method in interface itself	It is a method with default keyword and class can override this method
2.	Static method can invoke only on interface class not on class.	It can be invoked on interface as well as class
3.	Interface and implementing class, both can have static method with the same name without overriding each other.	We can override the default method in implementing class
4.	It can be used as a utility method.	It can be used to provide common functionality in all implementing classes