Java Vs C

Java	**C**
Java is Object Oriented Language	C is Procedural Oriented Language
Java is Interpreted Language	C is Compiled Language
High Level Language	Low Level Language
Buttom up approach	Top down approach
The Behind-the-scenes Memory Management with JAVA	The User-based Memory management in C
Platform independent	Platform dependent.
Pointer go backstage in JAVA	C requires explicit handling of pointers.

POP VS OOP

Comparison	POP	OOP
Approach	Top-down.	Bottom-up.
Basic	Main focus is on "how to get the task done" i.e. on the procedure or structure of a program	Main focus is on 'data security'. Hence, only objects are permitted to access the entities of a class.
Division	Large program is divided into units called functions.	Entire program is divided into objects.
Entity accessing mode	No access specifier observed.	Access specifier are "public", "private", "protected".
Overloading/Polymorphism	Neither it overload functions nor operators.	It overloads functions, constructors, and operators
Inheritance	Their is no provision of inheritance.	Inheritance achieved in three modes public private and protected.
Data hiding & security	There is no proper way of hiding the data, so data is insecure	Data is hidden in three modes public, private, and protected. hence data security increases.

Data sharing	Global data is shared among the functions in the program	Data is shared among the objects through the member functions.
Example	C, VB, FORTRAN, Pascal	C++, JAVA, VB.NET, C#.NET.

While Vs Do While

While	Do While
In 'while' loop the controlling condition appears at the start of the loop.	In 'do-while' loop the controlling condition appears at the end of the loop.
The iterations do not occur if the condition at the first iteration, appears false.	The iteration occurs at least once even if the condition is false at the first iteration.
Entry-controlled loop	Exit-controlled loop
Semi-colon Not used	Used at the end of the loop

Syntax : while (condition) {
statements; //body of loop
}
	Syntax : do{
	statements; // body of loop.
	} while(Condition);

Break Vs Continue

Break	Continue
A break can appear in both switch and loop (for, while, do) statements.	A continue can appear only in loop (for, while, do) statements.
A break causes the switch or loop statements to terminate the moment it is executed. Loop or switch ends abruptly when break is encountered.	A continue doesn't terminate the loop, it causes the loop to go to the next iteration. All iterations of the loop are executed even if continue is encountered. The continue statement is used to skip statements in the loop that appear after the continue.
The break statement can be used in both switch and loop statements.	The continue statement can appear only in loops. You will get an error if this appears in switch statement.
When a break statement is encountered, it terminates the block and gets the control out of the switch or loop.	When a continue statement is encountered, it gets the control to the next iteration of the loop.
A break causes the innermost enclosing loop	A continue inside a loop nested within a

or switch to be exited immediately.	switch causes the next loop iteration.

Constructor vs Method

Constructor	Method
Constructor is used to initialize an object	method is used to exhibit functionality of an object.
Constructors are invoked implicitly	methods are invoked explicitly.
Constructor does not return any value	the method may/may not return a value.
In case constructor is not present, a default constructor is provided by java compiler.	In the case of a method, no default method is provided.
Constructor should be of the same name as that of class	Method name should not be of the same name as that of class.

Static Vs Non Static Block

Static Block	Non Static Block
A static block is a block of code which contains code that executes at class loading time.	A non-static block executes when the object is created, before the constructor
A static keyword is prefixed before the start of the block.	There is no keyword prefix to make a block non-static block,unlike static blocks.
All static blocks executes only once per classloader	All non-static block executes every time an object of the containing class is created.

Syntax : static{
// code for static block
}
	Syntax:{
	// non static block
	}

Static Vs Non Static Method

Static Method	Non Static Method
A static method belongs to the class	a non-static method belongs to an object of a class.
Static methods are useful if you have only one instance where you're going to use the	Non-static methods are used if you're going to use your method to create multiple copies.

method, and you don't need multiple copies (objects).	
A static method can however be called both on the class as well as an object of the class. A static method can access only static members.	A non-static method can access both static and non-static members because at the time when the static method is called, the class might not be instantiated.
Syntax: public static void insert()	Syntax: public void start()

AutoBoxing and Unboxing

Autoboxing	Unboxing
Converting a primitive value into an object of the corresponding wrapper class is called autoboxing.	Converting an object of a wrapper type to its corresponding primitive value is called unboxing.
Eg: byte i1 = 1; Byte i = new Byte(i1);	Eg: Byte i = new Byte(1); byte i1 = i;

Check vs Unchecked Exception

Check	Uncheck
Check Exceptions are the exceptions that are checked at compile time.	Unchecked are the exceptions that are not 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.	if exceptions are unchecked, so it is not forced by the compiler to either handle or specify the exception. It is up to the programmers to be civilized, and specify or catch the exceptions.

Exception vs Error

Error : An Error “indicates serious problems that a reasonable application should not try to catch.” . Errors are the conditions which cannot get recovered by any handling techniques. It surely cause termination of the program abnormally. Errors belong to unchecked type and mostly occur at runtime. Some of the examples of errors are Out of memory error or a System crash error.

Eg:

```
class StackOverflow {
    public static void test(int i)
    {   //infinite loop
        if (i == 0)
            return;
        else
            test(i++);
    }
}
public class ErrorEg {
    public static void main(String[] args)
    {
        // eg of StackOverflowError
        StackOverflow.test(5);
    }
}
```

Exceptions : An Exception “indicates conditions that a reasonable application might want to catch.” Exceptions are the conditions that occur at runtime and may cause the termination of program. But they are recoverable using try, catch and throw keywords.

Eg:
```
public class ExceptionEg {
    public static void main(String[] args)
{       int a = 5, b = 0;
    // Attempting to divide by zero
    try {
            int c = a / b;
        }catch (ArithmeticException e) {
        e.printStackTrace();
    }
  }
}
```

Final Vs Finally Vs Finalize

final	finally	finalize
Final is used to apply restrictions on class, method and variable. Final class can't be inherited, final method can't be overridden and final variable value can't be	Finally is used to place important code, it will be executed whether exception is handled or not.	Finalize is used to perform clean up processing just before object is garbage collected.

changed.		
Final is a keyword.	Finally is a block	Finalize is a method.

Super vs This

Super()	This()
super() is use to call Base class's(Parent class's) constructor.	this() is used to call current class's constructor.
super() is required when there is inheritance So it can initialize the parent class.	this() is required when there is more than one constructor and that constructor is called inside another constructor
At Least two class is needed.	At least two constructor is need

class Parent {
Parent()
{
System.out.println("Parent class's No "+" arg constructor");
}
}
class Child extends Parent {
Child()
{
super();
System.out.println("Flow comes back from "+"Parent class no arg const");
}
public static void main(String[] args)
{
new Child();
System.out.println("Inside Main");
}
}
Output:
Parent class's No arg constructor
Flow comes back from Parent class no arg const
Inside Main
	class RR {
	RR()
	{
	this(10);
	System.out.println("Flow comes back from "+"RR class's 1 arg const");
	}
	
	RR(int a)
	{
	System.out.println("RR class's 1 arg const");
	}
	public static void main(String[] args)
	{
	new RR();
	System.out.println("Inside Main");
	}
	}
	
	Output:
	RR class's 1 arg const
	Flow comes back from RR class's 1 arg const
	Inside Main

This vs Super

this	super
this is used in the context of the class you are working on,	super is used to refer to the current instance of the parent class.

Eg :
class RR {
int a = 10;
static int b = 20;
void GFG()
{
this.a = 100;
System.out.println(a);
this.b = 600;
System.out.println(b);
}
 	
public static void main(String[] args)
{
new RR().GFG();
}
}
	Eg:
	class Parent {
	int a = 10;
	static int b = 20;
	}
	class Base extends Parent {
	void rr()
	{
	System.out.println(super.a);
	System.out.println(super.b);
	}
	
	public static void main(String[] args)
	{
	new Base().rr();
	}
	}