Java Basics

1) what are static blocks and instance block in Java ?

Static method:

- Static methods belong to the class and non-static methods belong to the objects of the class.
- A method declared with a static keyword is known as a static method. The static method belongs to a class instead of an instance/object. A static method is part of a class definition instead of part of the object.

When static variables and static methods are in the same class:

- A static method can access any static variable or static method without any object creation. To access a static variable or static method, we can use class names directly.
- But if the static variable and static method exist in the same class then we do not need to use the class name, Because of its presence in the same class. But to access a non-static variable or method we need to use the object.

Example:

```
class StaticMethodExample
{
  int a = 0;
  static int b = 0;

  void showData()
  {
    System.out.println("Value of a : "+ a);
    System.out.println("Value of b : "+ b);
  }

  static void displayData()
  {
```

```
    StaticMethodExample obj = new StaticMethodExample();
    System.out.println("Value of a : "+ obj.a);
    System.out.println("Value of b : "+ b);
  }


  public static void main(String arg[])
  {
    displayData();
    StaticMethodExample obj = new StaticMethodExample();
    obj.showData();
  }
}
```

When the static variable and static methods are in different classes:

- A static method can access any static variable or static method without any object creation. We can access them by use of the class names. But to access a non-static variable or method we need to use the object.

  Example:

```
class StaticExample
{
   int a = 0;
   static int b = 0;

   static String displayData()
   {
      return "Static method";
   }
}

public class MainClass
{
  public static void main(String arg[])
  {
```

```
        System.out.println("Value of b :"+ StaticExample.b);
        System.out.println(StaticExample.displayData());
    }
}
```

- We cannot use **this** and **super** keyword inside static method.
- We can't override the static method in java. If we declare a method with the same signature in the child class. It considered method hiding instead of method overriding.

  Example:

```
class ParentClass
{
  public static void printData()
  {
    System.out.println("Method of Parent class");
  }
}

public class ChildClass extends ParentClass
{
  public static void printData()
  {
    System.out.println("Method  of Child class");
  }

  public static void main(String arg[])
  {
    // Here we are creating object of Parent class but static method
always use class
    // So it will call the method of Parent class
    ParentClass parent = new ParentClass();
    parent.printData();

    // Here we are creating object of Child class but static method
always use class
    // So it will call the method of Child class
    ChildClass child = new ChildClass();
```

```
        child.printData();

        // Here we are creating object of Child class and reference of Parent
    class
        // But static method always use the class name
        // So it will call the method of Parent class
        ParentClass childObject = new ChildClass();
        childObject.printData();
      }
    }
```

- Static method supported in the interface

Static block:

- In Java, we can use the static keyword with a block of code that is known as a static block.
- A static block can have several instructions that always **run when a class is loaded into memory**.
- It is also known as java static initializer block because we can initialize the static variables in the static block at runtime.
- A class can have any number of static blocks, The JVM executes them in the sequence in which they have been written.
- The static block in a program is always executed first before any static method, non-static method, main method, or even instance block. Suppose we want to perform some operations at the time of class loading then we should use the static block.

NOTE: The static block always executes only one time when a class is loaded into memory.

```
static
{
  // Body of static block
}
```

- A class can have any number of static blocks that will execute when the class is loaded in memory.

- It is also called a static initialization block because it is used to initialize the variables at run time.

Static block and instance block (Non-static block):

- A class can have static blocks and non-static blocks(Instance blocks). Both sections execute before the constructor. But both have different purposes and execution times

NOTE: A non-static block doesn't get executed if we will not create an object. It gets invoked only when an object created by JVM.

Example:

```
public class MainClass
{
  static
  {
    System.out.println("Executing static block");
  }
  {
    System.out.println("Executing non-static block");
  }
  MainClass()
  {
    System.out.println("Executing Constructor");
  }


  public static void main(String arg[])
  {
    MainClass obj = new MainClass();
    System.out.println("Executing Main method");
  }
}
```

2) How to call one constructor from other constructors

- A constructor is called from another constructor in the same class this process is known as constructor chaining. It occurs through inheritance.
- When we create an instance of a derived class, all the constructors of the inherited class (base class) are first invoked, after that the constructor of the calling class (derived class) is invoked.

We can achieve constructor chaining in two ways:

- Within the same class: If the constructors belong to the same class, we use this
- From the base class: If the constructor belongs to different classes (parent and child classes), we use the super keyword to call the constructor from the base class.

The calling of the constructor can be done in two ways:

- By using this() keyword: It is used when we want to call the current class constructor within the same class.
- By using super() keyword: It is used when we want to call the superclass constructor from the base class.

Calling Current Class Constructor Example:

```
public class ConstructorChain
{
//default constructor
ConstructorChain()
{
this("Java");
System.out.println("Default constructor called.");
}
//parameterized constructor
ConstructorChain(String str)
{
System.out.println("Parameterized constructor called");
}
//main method
```

```
public static void main(String args[])
{
//initializes the instance of example class
ConstructorChain cc = new ConstructorChain();
}
}
```

Calling Super Class Constructor Example:

```
class Demo
{
//base class default constructor
Demo()
{
this(80, 90);
System.out.println("Base class default constructor called");
}
//base class parameterized constructor
Demo(int x, int y)
{
System.out.println("Base class parameterized constructor called");
}
}
//derived class or child class
class Prototype extends Demo
{
//derived class default constructor
Prototype()
{
this("Java", "Python");
System.out.println("Derived class default constructor called");
}
```

```
//derived class parameterized constructor
Prototype(String str1, String str2)
{
super();
System.out.println("Derived class parameterized constructor called");
}
}
public class ConstructorChaining
{
//main method
public static void main(String args[])
{
//initializes the instance of example class
Prototype my_example = new Prototype();
}
}
```

3) Method overriding in java

- 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.
- 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:

a. The method must have the same name as in the parent class
b. The method must have the same parameter as in the parent class.
c. There must be an IS-A relationship (inheritance).

Example:

```
class Bank{
int getRateOfInterest(){return 0;}
}
```

```
//Creating child classes.
class SBI extends Bank{
int getRateOfInterest(){return 8;}
}

class ICICI extends Bank{
int getRateOfInterest(){return 7;}
}
class AXIS extends Bank{
int getRateOfInterest(){return 9;}
}
//Test class to create objects and call the methods
class Test2{
public static void main(String args[]){
SBI s=new SBI();
ICICI i=new ICICI();
AXIS a=new AXIS();
System.out.println("SBI Rate of Interest: "+s.getRateOfInterest());
System.out.println("ICICI Rate of Interest: "+i.getRateOfInterest());
System.out.println("AXIS Rate of Interest: "+a.getRateOfInterest());
}
}
```

4) Super keyword in java

- 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:

a. super can be used to refer immediate parent class instance variable.
b. super can be used to invoke immediate parent class method.
c. super() can be used to invoke immediate parent class constructor.

a) 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.

Example:

```
class Animal{
String color="white";
}
class Dog extends Animal{
String color="black";
void printColor(){
System.out.println(color);//prints color of Dog class
System.out.println(super.color);//prints color of Animal class
}
}
class TestSuper1{
public static void main(String args[]){
Dog d=new Dog();
d.printColor();
}}
```

b) 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**.**

Example:

```
class Animal{
void eat(){System.out.println("eating...");}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");}
void bark(){System.out.println("barking...");}
void work(){
super.eat();
bark();
}
```

```
}
class TestSuper2{
public static void main(String args[]){
Dog d=new Dog();
d.work();
}}
```

c) super is used to invoke parent class constructor:

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

Example:

```
class Animal{
Animal(){System.out.println("animal is created");}
}
class Dog extends Animal{
Dog(){
super();
System.out.println("dog is created");
}
}
class TestSuper3{
public static void main(String args[]){
Dog d=new Dog();
}}
```

5) Difference between method overloading and method overriding in java

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.

6) Difference between abstract class and interface

- Abstract class and interface both are used to achieve abstraction where we can declare the abstract methods. Abstract class and interface both can't be instantiated.

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.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Example of abstract class and interface in Java

```
//Creating interface that has 4 methods
interface A{
void a();//bydefault, public and abstract
void b();
void c();
void d();
}

//Creating abstract class that provides the implementation of one method of A interface
abstract class B implements A{
public void c(){System.out.println("I am C");}
}
```

```
//Creating subclass of abstract class, now we need to provide the implementation of rest of the methods
class M extends B{
public void a(){System.out.println("I am a");}
public void b(){System.out.println("I am b");}
public void d(){System.out.println("I am d");}
}

//Creating a test class that calls the methods of A interface
class Test5{
public static void main(String args[]){
A a=new M();
a.a();
a.b();
a.c();
a.d();
}}
```

7) Why java is platform independent

- Unlike other programming languages, the bytecode produced by the javac compiler may be run on a variety of Operating Systems. In fact, Java's produced bytecode simply requires the JVM.
- The operating system has no impact on it. You can write and execute the code on any platform as long as the javac compiler is present.

8) What is method overloading in java

- If a class has multiple methods having same name but different in parameters, it is known as Method Overloading.
- Method overloading increases the readability of the program.

Different ways to overload the method

a. By changing number of arguments
b. By changing the data type

a. Method Overloading: changing no. of arguments:

```
class Adder{
static int add(int a,int b){return a+b;}
static int add(int a,int b,int c){return a+b+c;}
}
class TestOverloading1{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(11,11,11));
}}
```

b. Method Overloading: changing data type of arguments:

```
class Adder{
static int add(int a, int b){return a+b;}
static double add(double a, double b){return a+b;}
}
class TestOverloading2{
public static void main(String[] args){
System.out.println(Adder.add(11,11));
System.out.println(Adder.add(12.3,12.6));
}}
```

9) Difference between c++ and java:

Feature	C++	Java
Syntax	C++ has a complex syntax with pointers, multiple inheritance and templates.	Java has a simple, easy to learn syntax.
Performance	C++ is faster than Java because it is a compiled language.	Java is slower than C++ because it is an interpreted language.
Memory Management	C++ uses manual memory management.	Java uses automatic memory management.

Portability	C++ code can be recompiled on different platforms.	Java code can run on any platform that has a JVM installed.
Object Oriented	C++ is an Object Oriented language but does not have strict OOP concepts.	Java is a pure Object Oriented language.
Null Pointers	C++ has null pointers that can cause problems if not handled properly.	Java has no null pointers and automatically manages memory.
Platform Dependency	C++ is platform dependent and requires recompilation on different platforms.	Java is platform independent and can run on any platform with a JVM.
Type Checking	C++ has weak type checking and does not enforce strong typing.	Java has strong type checking and enforces strict typing.

10) What is JIT compiler

- The Just-In-Time (JIT) compiler is an essential part of the JRE i.e. Java Runtime Environment, that is responsible for performance optimization of java based applications at run time. The compiler is one of the key aspects in deciding the performance of an application for both parties i.e. the end-user and the application developer.
- In order to improve performance, JIT compilers interact with the Java Virtual Machine (JVM) at run time and compile suitable bytecode sequences into native machine code.
- While using a JIT compiler, the hardware is able to execute the native code, as compared to having the JVM interpret the same sequence of bytecode repeatedly and incurring overhead for the translation process. This subsequently leads to performance gains in the execution speed, unless the compiled methods are executed less frequently.
- The JIT compiler is able to perform certain simple optimizations while compiling a series of bytecode to native machine language. Some of these optimizations

performed by JIT compilers are data analysis, reduction of memory accesses by register allocation, translation from stack operations to register operations, elimination of common sub-expressions, etc.

- The greater is the degree of optimization done, the more time a JIT compiler spends in the execution stage. Therefore it cannot afford to do all the optimizations that a static compiler is capable of, because of the extra overhead added to the execution time and moreover it's view of the program is also restricted.

11) What is byte code in java

- 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. In more apt terms, java bytecode is the machine code in the form of a .class file. With the help of java bytecode we achieve platform independence in java.

12) Difference between this() and super():

- The 'this' and the 'super' keywords are reserved words that are used in a different context. Besides this, Java also provides this() and super() constructors that are used in the constructor context.

this()	**super()**
The this() constructor refers to the current class object.	The super() constructor refers immediate parent class object.
It is used for invoking the current class method.	It is used for invoking parent class methods.
It can be used anywhere in the parameterized constructor.	It is always the first line in the child class constructor.

13) What is a class ?

- In object-oriented programming, a class is a basic building block. It can be defined as template that describes the data and behaviour associated with the class instantiation. Instantiating is a class is to create an object (variable) of that class that can be used to access the member variables and methods of the class.
- A class can also be called a logical template to create the objects that share common properties and methods.

A class in Java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested class and interface

14) What is an object ?

- Object is a instance of a class and it is a parent class for all the classes in java.
- Object establishes the communication between the properties of the class.
- An entity that has state and behavior is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible). The example of an intangible object is the banking system.

An object has three characteristics:

a. State: represents the data (value) of an object.
b. Behavior: represents the behavior (functionality) of an object such as deposit, withdraw, etc.
c. Identity: An object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. However, it is used internally by the JVM to identify each object uniquely.

15) What is method in java ?

- A method is a block of code or collection of statements or a set of code grouped together to perform a certain task or operation.
- It is used to achieve the reusability of code. We write a method once and use it many times. We do not require to write code again and again. It also provides the easy modification and readability of code, just by adding or removing a chunk of code.
- The method is executed only when we call or invoke it.

16) What is encapsulation ?

- Encapsulation in Java is a process of wrapping code and data together into a single unit.
- We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.
- The **Java Bean** class is the example of a fully encapsulated class.

Advantage of Encapsulation in Java:

- By providing only a setter or getter method, you can make the class read-only or write-only.
- It provides you the control over the data. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.
- It is a way to achieve data hiding in Java because other class will not be able to access the data through the private data members.
- The encapsulate class is easy to test. So, it is better for unit testing.
- The standard IDE's are providing the facility to generate the getters and setters. So, it is easy and fast to create an encapsulated class in Java.

Example:

```
//A Account class which is a fully encapsulated class.
//It has a private data member and getter and setter methods.
class Account {
//private data members
private long acc_no;
```

```
private String name,email;
private float amount;
//public getter and setter methods
public long getAcc_no() {
   return acc_no;
}
public void setAcc_no(long acc_no) {
   this.acc_no = acc_no;
}
public String getName() {
   return name;
}
public void setName(String name) {
   this.name = name;
}
public String getEmail() {
   return email;
}
public void setEmail(String email) {
   this.email = email;
}
public float getAmount() {
   return amount;
}
public void setAmount(float amount) {
   this.amount = amount;
}

}
//A Java class to test the encapsulated class Account.
```

```
public class TestEncapsulation {
public static void main(String[] args) {
   //creating instance of Account class
   Account acc=new Account();
   //setting values through setter methods
   acc.setAcc_no(7560504000L);
   acc.setName("Sonoo Jaiswal");
   acc.setEmail("sonoojaiswal@gmail.com");
   acc.setAmount(500000f);
   //getting values through getter methods
   System.out.println(acc.getAcc_no()+" "+acc.getName()+"
"+acc.getEmail()+" "+acc.getAmount());
}
}
```

17) Why main() method is public, static and void in java / Explain about main() method in java

- In Java programs, the point from where the program starts its execution or simply the entry point of Java programs is the main() method. Hence, it is one of the most important methods of Java and having a proper understanding of it is very important.
- The Java compiler or JVM looks for the main method when it starts executing a Java program. The signature of the main method needs to be in a specific way for the JVM to recognize that method as its entry point. If we change the signature of the method, the program compiles but does not execute.

public static void main(String[] args)

Public:

It is an Access modifier, which specifies from where and who can access the method. Making the main() method public makes it globally available. 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 that is when associated with a method, making it a class-related method. The main() method is static so that JVM can invoke it without instantiating the class. This also saves the unnecessary wastage of memory which would have been used by the object declared only for calling the main() method by the JVM.

Void:

It is a keyword and is used to specify that a method doesn't return anything. As the main() method doesn't return anything, its return type is void. As soon as the main() method terminates, the java program terminates too. Hence, it doesn't make any sense to return from the main() method as JVM can't do anything with the return value of it.

main:

It is the name of the Java main method. It is the identifier that the **JVM** looks for as the starting point of the java program. It's not a keyword.

String[] args:

It stores Java command-line arguments and is an array of type java.lang.String class. Here, the name of the String array is args but it is not fixed and the user can use any name in place of it.

18) What is constructor in java ?

- In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
- It is a special type of method which is used to initialize the object.
- It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

Types:

I. No-arg constructor
II. Parameterized constructor

Rules for creating Java constructor:

a. Constructor name must be the same as its class name
b. A Constructor must have no explicit return type
c. A Java constructor cannot be abstract, static, final, and synchronized

Java Default Constructor:

- A constructor is called "Default Constructor" when it doesn't have any parameter.

Example of default constructor:

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Java Parameterized Constructor:

- A constructor which has a specific number of parameters is called a parameterized constructor.
- The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example of parameterized constructor:

```
class Student4{
  int id;
  String name;
  //creating a parameterized constructor
  Student4(int i,String n){
  id = i;
  name = n;
  }
  //method to display the values
  void display(){System.out.println(id+" "+name);}
```

```
  public static void main(String args[]){
  //creating objects and passing values
  Student4 s1 = new Student4(111,"Karan");
  Student4 s2 = new Student4(222,"Aryan");
  //calling method to display the values of object
  s1.display();
  s2.display();
 }
}
```

19) What is difference between length and length() method in java ?

- The length is an instance variable of an array in Java whereas length() is a method of String class.

  length:

  - An array is an object that holds a fixed number of values of the same type.
  - The length variable in an array returns the length of an array i.e. a number of elements stored in an array.
  - Once arrays are initialized, its length cannot be changed, so the length variable can directly be used to get the length of an array.
  - The length variable is used only for an array.

    Example:

```
public class ArrayLengthTest {
  public static void main(String args[]) {
    int array[] = {1, 2, 3, 4, 5, 6, 7};
    System.out.println("Length of an array is: " + array.length);
  }
}
```

  length():

  - The length() method is a static method of String class.
  - The length() returns the length of a string object i.e. the number of characters stored in an object.
  - String class uses this method because the length of a string can be modified using the various operations on an object.

- The String class internally uses a char[] array that it does not expose to the outside world.

  Example:

```
public class StringLengthMethodTest {
  public static void main(String args[]) {
    String str = "Welcome";
    System.out.println("Length of String using length() method is:
" + str.length());
  }
}
```

20) What is ASCII Code?

- ASCII stands for American Standard Code for Information Interchange. ASCII is a standard data-transmission code that is used by the computer for representing both the textual data and control characters.
- ASCII is a 7-bit character set having 128 characters, i.e., from 0 to 127. ASCII represents a numeric value for each character, such as 65 is a value of A.
- In Java, an ASCII table is a table that defines ASCII values for each character. It is also a small subset of Unicode because it contains 2 bytes while ASCII requires only one byte.

21) What is Unicode ?

- Unicode system is an international character encoding technique that can represent most of the languages around the world.
- Unicode System is established by Unicode Consortium.
- Hexadecimal values are used to represent Unicode characters.
- There are multiple Unicode Transformation Formats:
  - UTF-8: It represents 8-bits (1 byte) long character encoding.
  - UTF-16: It represents 16-bits (2 bytes) long character encoding
  - UTF-32: It represents 32-bits (4 bytes) long character encoding.
- To access a Unicode character the format starts with an escape sequence \u followed by 4 digits hexadecimal value.
- A Unicode character has a range of possible values starting from \u0000 to \uFFFF.
- Some of the Unicode characters are
  - \u00A9 represent the copyright symbol - ©

- \u0394 represent the capital Greek letter delta - Δ
- \u0022 represent a double quote - "

22) Difference between Character Constant and String Constant in java ?

- Character Constants are written by enclosing a character within a pair of single quotes. String Constants are written by enclosing a set of characters within a pair of double quotes.
- Character Constants are assigned to variables of type char. String Constants are assigned to variables of type String.

23) What are constants and how to create constants in java?

- Constant is a value that cannot be changed after assigning it. Java does not directly support the constants. There is an alternative way to define the constants in Java by using the non-access modifiers static and final.
- In Java, to declare any variable as constant, we use static and final modifiers. It is also known as non-access modifiers. According to the Java naming convention the identifier name must be in capital letters.
- The purpose to use the static modifier is to manage the memory.
- It also allows the variable to be available without loading any instance of the class in which it is defined.
- The final modifier represents that the value of the variable cannot be changed. It also makes the primitive data type immutable or unchangeable.

  Syntax:

  static final datatype identifier_name=value;

  Example:

  static final double PRICE=432.78;

24) Difference between '>>' and '>>>' operators in java:

>> Signed right shift

>>> Unsigned right shift

- The bit pattern is given by the left-hand operand, and the number of positions to shift by the right-hand operand. The unsigned right shift operator >>> shifts a zero into the leftmost position,
- while the leftmost position after >> depends on sign extension.

- In simple words >>> always shifts a zero into the leftmost position whereas >> shifts based on sign of the number i.e. 1 for negative number and 0 for positive number.

For example try with negative as well as positive numbers.

```
int c = -153;
System.out.printf("%32s%n",Integer.toBinaryString(c >>= 2));
System.out.printf("%32s%n",Integer.toBinaryString(c <<= 2));
System.out.printf("%32s%n",Integer.toBinaryString(c >>>= 2));
System.out.println(Integer.toBinaryString(c <<= 2));

System.out.println();

c = 153;
System.out.printf("%32s%n",Integer.toBinaryString(c >>= 2));
System.out.printf("%32s%n",Integer.toBinaryString(c <<= 2));
System.out.printf("%32s%n",Integer.toBinaryString(c >>>= 2));
System.out.printf("%32s%n",Integer.toBinaryString(c <<= 2));
```

Output:

```
11111111111111111111111111011001
11111111111111111111111101100100
  11111111111111111111111111011001
11111111111111111111111101100100

100110
10011000
100110
10011000
```