**Char Data Type**

The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.

**Example: char letterA = 'A'**

## Operator Precedence Chart in Java

| Operator | Precedence |
|---|---|
| Bracket | ( ) |
| Prefix increment ,decrement and unary | ++ -- + - ~ ! |
| Multiplicative | / * % |
| Addititive | + - |
| Postfix increment and decrement | ++ -- |
| Shift | << >> >>> |
| Relational | < > <= >= instance of |
| Equality | == != |
| Bitwise AND | & |
| Bitwise exclusive OR | ^ |
| Bitwise inclusive OR | \| |
| Bitwise AND | && |
| Bitwise OR | \|\| |
| Ternary | ? : |
| Assignment | = += -= *= /= %= &= ^= |

## MCQ Question

**Question1 :**

public class P {

```
   public static void main(String[] args) {

        int a = 10, b = 5, c = 1, result;
        result = a-++c-++b;
       //     result = a-(2)-(6)
       //     result = 10-2=8
       //     result = 8-6
       //     result = 2
        System.out.println (result);
   }
}
```

**Output: 2**

**Description of above code:**

If we think about the above code and think about the priority of operator then ++ operator having higher priority than – (binary minus) operator so ++ operator get executed before binary minus (- ) so your expression look like as after execution of ++ result=a-(++c)-(++b) means result=10-(2)-(6) after solve ++ operator we have the two binary minus operator present in equation so from left to right operator get executed means your equation get executed like as result=10-2-6 solve first result=8-6 solve later so final result is result=2.

**Question2:**

```
public class Second
{      public static void main(String x1[]) {
            int x = 0, y = 0 , z = 0 ;
            x = (++x + y-- ) * z++;
           System.out.println("X is  "+x);
    }
}
```

If we think about above code then its execution like as

Initially x=0,y=0;

x = (++x + y--) * z++;

++x means first it gets incremented later gets initialised //++x=1

y-- means it gets initialised first later gets decremented //y-- =-1

z++ means first it gets incremented later gets initialised //z++ =1
x=(1+ -1) * 1
x=0

**Question 3:**

```
class Numbers{
      public static void main(String args[]){
            int a=20, b=10;
            if((a < b) && (b++ < 25)){
                  System.out.println("This is any language logic");
            }
            System.out.println(b);
      }
}
```

**Description of above code**

If we think about the code initially value of a=10 and b=20 and when if statement get executed then (a<b) get executed the value of a is 20 and the value of b is 10 so the first condition get failed and we use the && operator in if condition so the rule of && is when first condition get failed then second condition not check by && so b++ not executed so the value of b is 10

**Question 4**

```
class MyClass
{
   public static void main(String s[])
   {  int i = 4;
      int j = 21;
      int k = ++i * 7 + 2 - j--;
      System.out.println("k = " + k+"\t J="+j);
   }
}
```

**Output: K= 16 J=20**

**Explanation of above code**

If we think about the above code we have the expression int k=++i *7+2-j—
Here we initialized the value i=4 and j=21 in this express we use the ++i this is the pre increment and pre increment having higher priority than * , + ,- and post increment so ++i get executed very first so after executing ++i your expression like as int k =5*7+2-j-- after so the ++i we have the * (mutliplicaton) is higher priority operator so * multiplication executed after multiplication our code is like as int k=35+2-j—after solve multiplication we have the + and – operator so + and – having same priority from left to right so + get executed first after + execution our equation is int k=37-j-- after solve + we have the two operator - and j- - Here j- - is post increment so post increment having less priority than arithmetic operator so minus (-) operator solve first so your equation like as int k= 37-21 so k=16 and after initialized k value j will decrement so the value of j is 20

**Example**

```
class IncDec
{  public static void main(String s[])
   {   int a = 1;
       int b = 2;
       int c;
       int d;

      c = ++b;
      d = a++;
      c++;
      System.out.println ("a = " + a);
      System.out.println ("b = " + b);
      System.out.println ("c = " + c);
      System.out.println ("d = " + d);
   }
}
```

**Output**

```
a = 2
b = 3
```

c = 4
d = 1

**Description of above code**

In above code a is initialized as 1 and b is initialized as 2 c and d is not initialized if we think about the statement c = ++b then it is pre incremented means first increase the value of b means c=3 and if we think about the statement d = a++ here a++ is post increment means first value of a is initialized in d means d=1 and then a++ executed means 1 increase by 1 from 1 to 2 so a=2 and again c++ means c is also increase by 1 means c was 3 so is c is increment by 1 so c=4
When we print the a it will print 2 when we print the b it will print 3 and when we print c it will print 4 and d is 1

**Example**

```
package org.techhub;
public class Demo
{
   public static void main(String[] args)
   {  int i, j, k, l = 0;
      k = l++;
      j = ++k;
      i = j++;
      System.out.println("I is  "+i);
   }
}
```

**Output**

```
I is  1
```

**Code Description**

If we think about the above code then we have the four variables i, j, k, l=0
If we think about k=l++ here we have the post increment so the initial value of l is 0 this value initialize in k first and after that l++ execute means after execution of this statement k=0 and l=1 and if we think about the j=++k statement here ++k is

pre increment statement means first ++k execute and then value initialized in j means k =1 first and then value initialized in j means j is also 1 and if we think about the i=j++ here j++ is post increment means first value of j is initialized in i and then j++ execute means the value of i=1 and j=2 so output is 1

**Example**

```
package org.techhub;
public class Demo {
   public static void main(String[] args) {
      int dailywage = 4;
      int number_of_days = 5;
     int salary;
salary = number_of_days++ * --dailywage * dailywage++ * number_of_days--;
      salary =- dailywage;
      System.out.println(dailywage + " " + number_of_days + " " + salary);
   }
}
```

**Output**

```
4 5 -4
```

**Description of above code**

If we think about the above code we have the three variables dailaywage=4 number_of_days=5 and salary

If we think about the statement salary=number_of_days++ * --dailywage *dailywage++ * number_of_days- - in this expression top most priority operator is - -dailywage and - - dailywage execute very first after executing - - dailywage your expression look like as

salary= number_of_days++ * dailywage * dailywage++ if we put the values in this expression then expression look like as salary=5 * 3 * 3 so we have the two operator in expression ++ and * but ++ in the form of post increment and post increment having less priority than * so first multiplication get executed and result of multiplication stored in salary so salary=45 after solve the multiplication ++ operator get executed number_of_days++ before increment

number_of_days=5 so it will increase by 1 and number_of_days=6 and after that dailywage++ execute so initially dailywage =4 but previous it is decrease by 1 with pre decrement operator so dailwage=3 at the time of multiplication but now dailywage++ execute so it will from 3 to 4 means now current value of dailywage=4 and after that number_of_days- - execute so number of days previous number_of_days=6 here again number_of_days decrease by 1 so it will again 5

and if we think about the statement salary = - dailywage so dailywage was 4 previous and when we initialize it in salary using negative sign then it will stored as -4 in salary and if we think about this statement System.***out***.println(dailywage + " " + number_of_days + " " + salary);

So we get the output 4 5 -4

**Example**

```
package org.techhub;
class Demo
{ public static void main(String s[])
   { int i = 34.0;
      int j = 7;

      int k = i % j;
     System.out.println("k = " + k );
   }
}
```

**Output: compile time error**

If we think about the above code we try to store int i=34.0 in integer and 34.0 is double value so we cannot store in integer directly so it will generate the compile time error at run time.

**Example**

```
package org.techhub;
class Demo
{
```

```
   public static void main(String s[])
   {
      int x = 42;
      double y = 42.25;
      System.out.println("x mod 10 = " + x % 10 );
      System.out.println("y mod 10 = " + y % 10 );
   }
}
```

**Output**

```
x mod 10 = 2
y mod 10 = 2.25
```

**Code description**

If we think about the above code we try to apply the % mod operator on double value but after jdk 1.7 version of java % (mod) can apply on floating value so the result is mention above

**Example**

**Which of the following statements is true?**

- **1. When a = 5 the value of a % 2 is same as a - 4**
- **2. When a = 3 the value of a * 3 * 3 is greater than ( a + 10 ) * 3**
- **3. When a = 7 the value of a * 7 * 3 is greater than ( a * a + 7 * a + 3 )**

Output: 1 and 3

**Code Description**

Statement 1: a = 5. a % 2 = 5 % 2 = 1anda - 4 = 5 - 4 = 1. Both values are equal. So statement 1 is true.

Statement 2: a = 3. a * 3 * 3 = 27 and ( a + 10 ) * 3 = ( 3 + 10 ) * 3 = 13 * 3 = 39. But 27 is not greater than 39. So statement 2 is false.

Statement 3: a = 7. a * 7 * 3 = 147 and ( a * a + 7 * a + 3 ) = ( 7 * 7 + 7 * 7 + 3 ) = ( 49 + 49 + 3 ) = 101. 147 is greater than 101. So statement 3 is true.

**Extra Tip**: Operator Precedence (highest to lowest)

* / % left to right (associativity)
+ - left to right (associativity)

**Example**

```
package org.techhub;
public class Demo {
   public static void main(String[] args) {
   System.out.println(10 * 5 + 100 * (25 * 11) / (25 * 10) * 10 - 5 + 7 % 2);
   int zx = (10 * 5 + 100 * (25 * 11));
   int yz = ((25 * 10) * 10 - 5 + 7 % 2);
   System.out.println(zx / yz);
   }
}
```

**Output**

```
1146
11
```

Example

```
package org.techhub;
class Demo{
   public static void main(String args[])
   {
      int var1 = 42;
      int var2 = ~var1;
      System.out.print(var1 + " " + var2);
   }
}
```

Output

```
42 -43
```

**Code Description**

Unary not operator, ~, inverts all of the bits of its operand. 42 in binary is 00101010 in using ~ operator on var1 and assigning it to var2 we get inverted value of 42 i:e 11010101 which is -43 in decimal.

**Example**

```
package org.techhub;
class Demo {
        public static void main(String args[]) {
                int a = 3;
                int b = 6;
                int c = a | b;
                int d = a & b;
                System.out.println(c + "\t" + d);
        }
}
```

Output

```
7       2
```

And operator produces 1 bit if both operand are 1. Or operator produces 1 bit if any bit of the two operands in 1.

Example

```
package org.techhub;
class Demo {
   public static void main(String args[])
   {
      byte x = 64;
      int i;
      byte y;
      i = x << 2;
      y = (byte) (x << 2);
      System.out.print(i + "\t" + y);
   }
```

```
}
```

Output

```
256 0
```

Example

```
package org.techhub;
class Demo {
   public static void main(String args[])
   {
      int x;
      x = 10;
      x = x >> 1;
      System.out.println(x);
   }
}
```

Output

```
5
```

Example

```
package org.techhub;
class Demo {
   public static void main(String args[])
   {
      int a = 1;
      int b = 2;
      int c = 3;
      a |= 4;
      b >>= 1;
      c <<= 1;
      a ^= c;
      System.out.println(a + " " + b + " " + c);
   }
```

```
}
```

Output

```
3  1  6
```

Example

```
public class UnaryOperators {
	public static void main(String[] args) {
		int i = 5;
		System.out.print(i++);
		System.out.print(++i);
		System.out.print(i--);
		System.out.print(--i);
	}
}
```

Output

```
 5775
```

**Example**

```
public class ShiftOperators {
	public static void main(String[] args) {
		System.out.print(8<<2);
		System.out.print(",");
		System.out.print(8>>1);
	}
}
```

Output:

```
32, 4
```

**Example**

```
public class BitwiseNotOperator{
	public static void main(String[] args) {
		int i = 50;
		System.out.print(~i);
		System.out.print(",");
```

```
            System.out.print(~--i);
            System.out.print(",");
            System.out.print(~++i);
        }
}
```

Output

**-51,-50,-51**

**Example**

```
public class RelationalOperators {
        public static void main(String[] args) {
            int a = 4;
            int b = 5;
            System.out.print(a>b);
            System.out.print(",");
            System.out.print(a<b);
        }
}
```

Output

**false , true**

**Example**

```
public class LogicalOperators {
        public static void main(String[] args) {
            int a = 5;
            int b = 10;
            boolean flag = a>b;
            System.out.println(!flag);
        }
}
```

**Output**

True

**Example**

```
public class TernaryOperators {
	public static void main(String[] args) {
		int a = 10;
		String result = a > 5 ? "Hello 1" : "Hello 2";
		System.out.println(result);
	}
}
```

**Output**

**Hello 1**

**Example**

```
class Ques
{
 public static void main(String args[])
 {
    byte b = 12;
    int y = b;
    b = b + 10;
    System.out.println(b);
  }
}
```

**Output**

The program will lead to compile time error as explicit casting is required in the line, b = b + 10.

**Explanation**

When you compile the above program, a compile time error occurs in the line, b = b + 10 as explicit casting is required to assign an int value to a byte type. In the preceding program the byte type variable is declared and initialized to the value 12. Then the value of byte variable is assigned to the int variable, y as assigning byte type value to an int variable is possible. However the compilation error of

loss of precision will occur while incrementing the byte type value with 10, i.e. int value. Therefore A, C, and D are incorrect options

**Example**

**Which of the following are valid declarations of the main () method?**

A. static main(String args[]){ }

B. public static String main(String args[]) {… }

C. public static void main(String args[]) {….}

D. final static void main(String args[]) {….}

**Output:**

The correct option is C.

**Explanation:** The following is a valid declaration of the static method: public static void main(String args[]) { //implementation of the main method }

Therefore, the correct option is C since the return type of the main method is void and is declared as static

**Example**

**Ram as a developer was asked to create a program using switch…case within for loop. Ram created the following program:**

```
class Ram {
public static void main(String args[])
{
   int z=3;
   for(int i=0; i<2;i++)
    {
   z++;
    switch(z)
    {
      case 3:
          System.out.print(z=z+1 +" ");
```

```
    case 5:
         System.out.print(z=z+2 + " ");
        break;
    default :
         System.out.print(z=z+8 + " ");
    case 6:
         System.out.print( z=z+4 + " ");
     }
    z--;
   }
 }
}
```

**Output**

12 16 24 28

**Explanation:** Option D is correct Initially the variable z is initialized with 3, which becomes 4 inside the for loop therefore, the cases before default becomes false and default statement prints 12 then statement following default is executed because break is not used after default and therefore 16 will be printed. Then, the value of z decreases by 1 and becomes 15. Now, for loop executes once again and z becomes 16 and same process continued and z becomes 24 in default case and 28 in next case. Option B is incorrect because for loop is executed two times. Option A is incorrect because value of z is 4 and none of the case statement is 4 and same is the case in option C.

**Example**

Imagine, you as a student provided with the following program during a class test

```
class Student {
 public static void main(String args[]){
 int z=6, k;
 for(int i=0; i<2;i++) {
 z++;
```

```
switch(z) {
case 3: System.out.print(z=z+1 + " ");
case 5: System.out.print(z=z+2 + " ");
break;
default : {
for (int x=10; x>3; x++) {
System.out.print(x=k+x + " ");
}
}
case 6: System.out.print( z=z+4 + " ");
}
z--;
}
}
}
```

**What would be the output of the preceding program?**

A. Program will display 8 10 as an output

B. Program will not compile successfully

C. Program runs infinity endless

D. Program will display 8 10 10as an output

**Output:** Option B is correct.

**Explanation:** Option B is correct. Program will not compile because k is not initialized. Option C will be correct when k is initialized but this time it is incorrect. Options A and D are incorrect because the program will not compile successfully.

**Example**

---

**Ram as a student was provided with the following code snippet**

```
public void Ram(float c) {
switch (c) {
case 5:
case 7:
case 2:
```

```
 default:
 case 9.5:
 }
}
```

After viewing the code snippet Ram was asked to notice the problems in preceding code snippet on the basis of the rules regarding switch…case statement. Following are the options from which Ram has to choose the correct answer.

A. There is no problem in the code snippet
B. Switch cannot evaluate float value

C. The default statement cannot be used between case statements
D. All cases must be in increasing order

**Output**

Option B is correct.

**Explanation:** Option B is correct because switch…case statement can evaluate to a char, byte, short, int, or enum. Therefore Option A is incorrect because float value is being used as a case, which is not allowed. Option C is incorrect because default can be used anywhere in switch…case block. Option D is incorrect because it is not necessary that cases must be in increasing order.

**Example**

Sheela as a faculty given following options to her students and asked them to choose the correct options:

A. A switch statement can only evaluate to float and double values
B. A switch…case block must have break statements after every case
C. Switch case must be similar to switch expression type
D. A switch…case can be nested like nested if…else

**Option D is correct.**

**Explanation:** Option D is correct because it is a fact.
Option A is incorrect because a switch…case cannot evaluate float and double. Option B is incorrect because there is no need to have break statement after every case. Option C is incorrect, for example, your case expression is of char type but you used 65 as case label then internally that 65 is recognized as char A. Therefore, case expression and case label can be varied but must take attention before using them

**Example**

Shyam during an interview was provided with following code and asked to review the program

```
public class Sam
 {
 public static void main(String args[])
 {
   int x=0, i=0;
   for (int y=0; y>=i; ++y,i++) {
     System.out.println(y);
    System.out.println(i);
    }
  }
}
```

**After reviewing the code he was asked to predict the correct options among the following:**

A. Program will print 0 0 for first time
B. Program results in an endless loop
C. Program will not compile because declaration is not allowed inside the for loop
D. Program will successfully compile and print 0 0 on execution and then terminates

**Option B is correct.**

**Explanation:** Option B is correct because the value of y will always be equals to i and therefore loop will never terminates so, option A is incorrect.
Option C is incorrect because declaration can be done inside the for loop.
Option D is incorrect because program will successfully compile but when executes it becomes an endless loop.

**Example**

Rani during an interview was shown the following program:

```
class Rani {
 public static void main(String args[]) {
 int x = 0;int y=9;
 for ( ; x<y; ) { x++; y++;} // (a)
 for (x; x==y; --x) continue; // (b)
 for (x=0; x<5; ) { x++; } // (c)
 for ( ; ; ) ; // (d)
 }
}
```

What would be the output of the preceding program from the following options?
A. Program will successfully compile and executes but does not print any value
B. Program will successfully compile and becomes endless because of loop d
C. Program will not compile because loop b is syntactically incorrect
D. Program will not compile because loop a has only expression part but missing initialization and Increment/decrement part

**Option C is the correct answer.**

**Explanation:** Correct answer is option C because the loop is not initialized. Therefore, options A and B are incorrect. Option D is incorrect because syntax of loop is correct.

**Example**

Shyam was given the following code snippet during an interview and asked to choose all correct decisional and loop statements:

```
int y=9;
for ( ;true ; ) { break;} // 1
if(y==9) { break; } // 2
switch(y) {default: break;} // 3
do ( ){ // code } while(expression); // 4
while ( ) { //code } // 5
```

**Options:**

A. Statement 1 and 3 are correct B. Statement 1, 2, and 3 are correct
C. Statement 1, 3, and 5 are correct D. Statement 1,4, and 5 are correct
E. Statement 1, 2, and 4 are correct

**Correct option is A.**

**Explanation:** Correct option is A. Reason for all other options can be verified on the basis of following facts:
The break statement can only be used in looping constructs and if need to use with if then if must be inside a loop the do… while do not have expression part with do i.e. do ( ) .The expression is used in while block. The while block Cannot be used without specifying expression.

**Example**

---

Shyam was given the following program by his teacher

```
class Sam {
 public static void main(String args[])
{
 int y=2;int i;
 for (i=0; i <= 3; i++) {
    if (i == 2) {
      break;
   }
   else
```

```
   {
      y++;
    }
 }
 System.out.println(i + ", " + y);
 }
}
```

**What would be the output of the preceding code?**

A. Program will display 2 , 2 as an output
B. Program will display 2 , 3 as an output
C. Program will display 2 , 4 as an output
D. Program will display 1 , 2 as an output

**Option C is the correct.**

**Explanation:** Option C is the correct output. Inside the for loop condition i==2 is checked and when it Evaluates to false the value of y will increment by 1. This process continues until i is not equal to 2. When the condition i==2 evaluates to true then the loop terminates and the vales of i and y are printed.
Rest all of the options are incorrect as they are representing incorrect output.

**Example**

---

Sheela after attending a lecture on for statement was shown the following for statements to choose the correct for statement:

A. for( int j=2; j*j==4, j<4; j++)  B.  for (int j=3; j/2==1; j++)
C. for (int j=3, long k=0; j>k; j++)  D.  int k, j; for (j=3, k=2; k==j-1; k++, j--)

**Options B and D are correct for statements.**

**Explanation:** Options B and D are correct for statements.
Option A is incorrect because multiple conditions cannot be used in for statement.  Option C is incorrect because different type of initialization variable

cannot be declared inside the for loop rather they can be declared outside the for loop.

**Example**

**Shyam works in a xyz company and he designed the following program:**

```
class Shyam {
 public static void main (String args[]) {
    int x=2; int y=6;
   if( x!=y || (y*=x)!=x) {
      System.out.println(" Not equal");
  }
  else{
   System.out.println(" Equal");
   }
 }
}
```

What happens when he compile and run the preceding program?

A. Program will display Equal

B. Program will display Not Equal

C. Program will not compile successfully because if statement is not correct

D. Program will compile but not executes

**Option B is the correct answer.**

**Explanation:** Option B is correct because the first expression in if statement evaluates to true therefore Second expression is not checked and the statement in else block is displayed. Option A is incorrect because if statement evaluates to false. Options C and D are incorrect because the program compiles and executes successfully

**Example**

**Rems and Sam while preparing for Java certification created the following program:**

```
public class Rose{
 public static void main(String[] args)
{
   char x = 'a';
   switch(x)
    {
      case 66: System.out.println( "B" +" ");break;
      case 72: System.out.println( "H"+ " ");break;
      case 97: System.out.println("a"+ " +);
      case 89: System.out.println( "Y" + " ");break;
     default: System.out.println( "default");break;
   }
 }
}
```

**What would be the output of this program? Choose the correct option from the following options:**

A. Program will display a  Y default

B. Program will display a Y

C. Program will not compile successfully because break cannot be used with default case.

D. Program will display A

**Option B is the correct answer.**

**Explanation:**

Option B is correct because ASCII equivalent of small a is 97 and because break is not used so, statement following this case is also displayed. Option A is incorrect because break statement is used in case 89 (ASCII equivalent of Y).

Option C is incorrect because break can be used with default statement.

Option D is incorrect because ASCII equivalent of capital A is 65, which is not a case in this program.