CHP-7
Java Modifier Types

Modifiers arekeywords that you add to those definitions to change their meanings. The Java language has a wide variety of modifiers, including the following:

1. Java Access Modifiers

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are:

- Visible to the package, the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).

Default Access Modifier --- No keyword:

Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.
A variable or method declared without any access control modifier is available to any other class in the same package. The fields in an interface are implicitly public static final and the methods in an interface are by default public

Example:

Variables and methods can be declared without any modifiers, as in the following examples:

```
String version ="1.5.1";
boolean processOrder(){
return true;
}
```


Private Access Modifier --- private:

Methods, Variables and Constructors that are declared private can only be accessed within the declared class itself.
Private access modifier is the most restrictive access level. Class and interfaces cannot be private.
Variables that are declared private can be accessed outside the class if public getter methods are present in the class.
Using the private modifier is the main way that an object encapsulates itself and hide data from the outside world.

Example:

The following class uses private access control:

```
public class Logger{
private String format;
public String getFormat(){
```

```
return this.format;
}
public void setFormat(String format){
this.format = format;
}
}
```

Here, the *format* variable of the Logger class is private, so there's no way for other classes to retrieve or set its value directly.
So to make this variable available to the outside world, we defined two public methods: *getFormat()*, which returns the value of format, and *setFormat(String)*, which sets its value.

Public Access Modifier --- public:

A class, method, constructor, interface etc declared public can be accessed from any other class. Therefore fields, methods, blocks declared inside a public class can be accessed from any class belonging to the Java Universe. However if the public class we are trying to access is in a different package, then the public class still need to be imported.
Because of class inheritance, all public methods and variables of a class are inherited by its subclasses.

Example:

The following function uses public access control:

```
public static void main(String[] arguments){
// ...
}
```

The main() method of an application has to be public. Otherwise, it could not be called by a Java interpreter (such as java) to run the class.

Protected Access Modifier --- protected:

Variables, methods and constructors which are declared protected in a superclass can be accessed only by the subclasses in other package or any class within the package of the protected members' class.
The protected access modifier cannot be applied to class and interfaces. Methods, fields can be declared protected, however methods and fields in a interface cannot be declared protected.
Protected access gives the subclass a chance to use the helper method or variable, while preventing a nonrelated class from trying to use it.

Example:

The following parent class uses protected access control, to allow its child class override*openSpeaker()* method:

```
class AudioPlayer{
protected boolean openSpeaker(Speaker sp){
// implementation details
}
}
c
lass StreamingAudioPlayer{
boolean openSpeaker(Speaker sp){
// implementation details
}
}
```

Here, if we define *openSpeaker()* method as private, then it would not be accessible from any other class other than *AudioPlayer*. If we define it as public, then it would become accessible to all the outside world. But our intension is to expose this method to its subclass only, thats why we used*protected* modifier.

Access Control and Inheritance:

The following rules for inherited methods are enforced:
- Methods declared public in a superclass also must be public in all subclasses.
- Methods declared protected in a superclass must either be protected or public in subclasses; they cannot be private.
- Methods declared without access control (no modifier was used) can be declared more private in subclasses.
- Methods declared private are not inherited at all, so there is no rule for them.

2. Non Access Modifiers

To use a modifier, you include its keyword in the definition of a class, method, or variable. The modifier precedes the rest of the statement, as in the following examples (Italic ones):

```
public class className {
// ...
}
private boolean myFlag;
static final double weeks =9.5;
protected static final int BOXWIDTH =42;
public static void main(String[] arguments){
```



```
// body of method
}
```

Access Control Modifiers:

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are:

- Visible to the package. the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).

Non Access Modifiers:

Java provides a number of non-access modifiers to achieve many other functionality.

- The *static* modifier for creating class methods and variables
- The *final* modifier for finalizing the implementations of classes, methods, and variables.
- The *abstract* modifier for creating abstract classes and methods.
- The *synchronized* and *volatile* modifiers, which are used for threads.

To use a modifier, you include its keyword in the definition of a class, method, or variable. The modifier precedes the rest of the statement, as in the following examples (Italic ones):

```
publicclass className {
// ...
}
private boolean myFlag;
static final double weeks =9.5;
protected static final int BOXWIDTH =42;
public static void main(String[] arguments){
// body of method
}
```

Access Control Modifiers:

Java provides a number of access modifiers to set access levels for classes, variables, methods and constructors. The four access levels are:

- Visible to the package. the default. No modifiers are needed.
- Visible to the class only (private).
- Visible to the world (public).
- Visible to the package and all subclasses (protected).

Non Access Modifiers:

Java provides a number of non-access modifiers to achieve many other functionality.

- The *static* modifier for creating class methods and variables
- The *final* modifier for finalizing the implementations of classes, methods, and variables.
- The *abstract* modifier for creating abstract classes and methods.
- The *synchronized* and *volatile* modifiers, which are used for threads.

What is Next?

In the next section, I will be discussing about Basic Operators used in the Java Language. The chapter will give you

an overview of how these operators can be used during application developm**ent.**

CHP-8
Java Basic Operators

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Misc Operators

The Arithmetic Operators:

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators:

Assume integer variable A holds 10 and variable B holds 20, then:

Operator Description Example

+ Addition - Adds values on either side of the operator A + B will give 30
- Subtraction - Subtracts right hand operand from left hand operand A - B will give -10
* Multiplication - Multiplies values on either side of the operator A * B will give 200
/ Division - Divides left hand operand by right hand operand B / A will give 2
% Modulus - Divides left hand operand by right hand operand and returns remainder B % A will give 0
++ Increment - Increases the value of operand by 1 B++ gives 21
-- Decrement - Decreases the value of operand by 1 B-- gives 19

CHAPTER

TUTORIALS POINT Simply Easy Learning

Example

The following simple example program demonstrates the arithmetic operators. Copy and paste the following Java program in Test.java file and compile and run this program:

```
public class Test{
public static void main(String args[]){
int a =10;
int b =20;
int c =25;
int d =25;
System.out.println("a + b = "+(a + b));
```

```
System.out.println("a - b = "+(a - b));
System.out.println("a * b = "+(a * b));
System.out.println("b / a = "+(b / a));
System.out.println("b % a = "+(b % a));
System.out.println("c % a = "+(c % a));
System.out.println("a++ = "+(a++));
System.out.println("b-- = "+(a--));
// Check the difference in d++ and ++d
System.out.println("d++ = "+(d++));
System.out.println("++d = "+(++d));
}
}
```

This would produce the following result:

```
a + b =30
a - b =-10
a * b =200
b / a =2
b % a =0
c % a =5
a++=10
b--=11
d++=25
++d =27
```

The Relational Operators:

There are following relational operators supported by Java language:
Assume variable A holds 10 and variable B holds 20, then:

Operator Description Example

== Checks if the values of two operands are equal or not, if yes then condition becomes true. (A == B) is not true.

!= Checks if the values of two operands are equal or not, if values are not equal then condition becomes true. (A != B) is true.

> Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true. (A > B) is not true.

< Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true. (A < B) is true.

>= Checks if the value of left operand is greater than or equal to the value (A >= B) is not true.

of right operand, if yes then condition becomes true.

<= Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true. (A <= B) is true.

Example

The following simple example program demonstrates the relational operators. Copy and paste the following Java program in Test.java file and compile and run this program. :

```
public class Test{
public static void main(String args[]){
int a =10;
int b =20;
System.out.println("a == b = "+(a == b));
System.out.println("a != b = "+(a != b));
System.out.println("a > b = "+(a > b));
System.out.println("a < b = "+(a < b));
System.out.println("b >= a = "+(b >= a));
System.out.println("b <= a = "+(b <= a));
}
}
```

This would produce the following result:

```
a == b =false
a != b =true
a > b =false
a < b =true
```

```
b >= a =true
b <= a =false
```

The Bitwise Operators:

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte. Bitwise operator works on bits and performsbit-by-bit operation. Assume if a = 60; and b = 13; now in binary format they will be as follows:

a = 0011 1100
b = 0000 1101
---
a&b = 0000 1100
a|b = 0011 1101
a^b = 0011 0001
~a = 1100 0011

The following table lists the bitwise operators:

Assume integer variable A holds 60 and variable B holds 13, then:

Operator Description Example

& Binary AND Operator copies a bit to the result if it exists in both operands. (A & B) will give 12 which is 0000 1100

| Binary OR Operator copies a bit if it exists in either operand. (A | B) will give 61 which is 0011 1101

^ Binary XOR Operator copies the bit if it is set in one operand but not both. (A ^ B) will give 49 which is 0011 0001

~ Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. (~A) will give -60 which is 1100 0011

<<
Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.
A << 2 will give 240 which is 1111 0000

>>
Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.
A >> 2 will give 15 which is 1111

>>>
Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.
A >>>2 will give 15 which is 0000 1111

Example

The following simple example program demonstrates the bitwise operators. Copy and paste the following Java program in Test.java file and compile and run this program:

```
public class Test{
public static void main(String args[]){
int a =60; /* 60 = 0011 1100 */
int b =13; /* 13 = 0000 1101 */
int c =0;
c = a & b;/* 12 = 0000 1100 */
System.out.println("a & b = "+ c );
c = a | b;/* 61 = 0011 1101 */
System.out.println("a | b = "+ c );
c = a ^ b;/* 49 = 0011 0001 */
System.out.println("a ^ b = "+ c );
c =~a;/*-61 = 1100 0011 */
System.out.println("~a = "+ c );
c = a <<2;/* 240 = 1111 0000 */
System.out.println("a << 2 = "+ c );
c = a >>2;/* 215 = 1111 */
```

```
System.out.println("a >> 2 = "+ c );
c = a >>>2;/* 215 = 0000 1111 */
System.out.println("a >>> 2 = "+ c );
}
```



```
}
```

This would produce the following result:

```
a & b =12
a | b =61
a ^ b =49
~a =-61
a <<2=240
a >>15
a >>>15
```

The Logical Operators:

The following table lists the logical operators:

Assume Boolean variables A holds true and variable B holds false, then:

Operator Description Example

&& Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. (A && B) is false.

|| Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true. (A || B) is true.

! Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false. !(A && B) is true.

Example

The following simple example program demonstrates the logical operators. Copy and paste the following Java program in Test.java file and compile and run this program:

```
public class Test{
public static void main(String args[]){
boolean a =true;
boolean b =false;
System.out.println("a && b = "+(a&&b));
System.out.println("a || b = "+(a||b));
System.out.println("!(a && b) = "+!(a && b));
}
}
```

This would produce the following result:

```
a && b =false
a || b =true
!(a && b)=true
```

The Assignment Operators:

There are following assignment operators supported by Java language:

Operator Description Example

= Simple assignment operator, Assigns values from right side operands to left side operand C = A + B will assign value of A + B into C

+=

Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand

C += A is equivalent to C = C + A

-=

Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand

C -= A is equivalent to C = C - A

*=

Multiply AND assignment operator, It multiplies

right operand with the left operand and assign the result to left operand
C *= A is equivalent to C = C * A
/=
Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand
C /= A is equivalent to C = C / A
%=
Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand
C %= A is equivalent to C = C % A
<<= Left shift AND assignment operator C <<= 2 is same as C = C << 2
>>= Right shift AND assignment operator C >>= 2 is same as C = C >> 2
&= Bitwise AND assignment operator C &= 2 is same as C = C & 2
^= bitwise exclusive OR and assignment operator C ^= 2 is same as C = C ^ 2
|= bitwise inclusive OR and assignment operator C |= 2 is same as C = C | 2

Example:

The following simple example program demonstrates the assignment operators. Copy and paste the following Java program in Test.java file and compile and run this program:

```
public class Test{
public static void main(String args[]){
int a =10;
int b =20;
int c =0;
c = a + b;
System.out.println("c = a + b = "+ c );
c += a ;
System.out.println("c += a = "+ c );
c -= a ;
System.out.println("c -= a = "+ c );
c *= a ;
System.out.println("c *= a = "+ c );
a =10;
c =15;
```



```
c /= a ;
System.out.println("c /= a = "+ c );
a =10;
c =15;
c %= a ;
System.out.println("c %= a = "+ c );
c <<=2;
System.out.println("c <<= 2 = "+ c );
c >>=2;
System.out.println("c >>= 2 = "+ c );
c >>=2;
System.out.println("c >>= a = "+ c );
c &= a ;
System.out.println("c &= 2 = "+ c );
c ^= a ;
System.out.println("c ^= a = "+ c );
c |= a ;
System.out.println("c |= a = "+ c );
}
}
```

This would produce the following result:

```
c = a + b =30
c += a =40
c -= a =30
```

```
c *= a =300
c /= a =1
c %= a =5
c <<=2=20
c >>=2=5
c >>=2=1
c &= a =0
c ^= a =10
c |= a =10
```

Misc Operators

There are few other operators supported by Java Language.

Conditional Operator (?:):

Conditional operator is also known as the ternary operator. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide which value should be assigned to the variable. The operator is written as:

```
variable x =(expression)? value iftrue: value iffalse
```

Following is the example:

```
public class Test{
```



```
public static void main(String args[]){
int a , b;
a =10;
b =(a ==1)?20:30;
System.out.println("Value of b is : "+ b );
b =(a ==10)?20:30;
System.out.println("Value of b is : "+ b );
}
}
```

This would produce the following result:

```
Value of b is:30
Value of b is:20
```

instanceof Operator:

This operator is used only for object reference variables. The operator checks whether the object is of a particular type(class type or interface type). instanceof operator is wriiten as:

```
(Object reference variable ) instanceof (class/interface type)
```

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is the example:

```
String name = "James";
boolean result = name instanceof String;
// This will return true since name is type of String
```

This operator will still return true if the object being compared is the assignment compatible with the type on the right. Following is one more example:

```
classVehicle{}
public class CarextendsVehicle{
public static void main(String args[]){
Vehicle a =newCar();
boolean result = a instanceofCar;
System.out.println(result);
}
}
```

This would produce the following result:

```
true
```

Precedence of Java Operators:

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it

first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category Operator Associativity

Postfix () [] . (dot operator) Left to right
Unary ++ - - ! ~ Right to left
Multiplicative * / % Left to right
Additive + - Left to right
Shift >>>>><< Left to right
Relational >>= <<= Left to right
Equality == != Left to right
Bitwise AND & Left to right
Bitwise XOR ^ Left to right
Bitwise OR | Left to right
Logical AND && Left to right
Logical OR || Left to right
Conditional ?: Right to left
Assignment = += -= *= /= %= >>= <<= &= ^= |= Right to left
Comma , Left to right

What is Next?

Next chapter would explain about loop control in Java programming. The chapter will describe various types of loops and how these loops can be used in Java program development and for what purposes they are being used.