Chapter 9

Methods and Types of Methods

- In this chapter, You will learn
  - Definition of method
  - Method terminology
  - Types of methods
  - Rules in calling different methods
  - Methods execution control flow
  - Modifiers allowed for a method
- By the end of this chapter- you will learn defining, declaring, calling methods, and its logic execution control flow.

i

Interview Questions

By the end of this chapter you answer all below interview questions

- Method definition
- Method terminology
  - Method prototype
  - Method body and Logic
  - Parameters and arguments
  - Method Signature
  - Method return type
  - Method Modifier
- Main method's terminology
- Defining, declaring and invoking/calling a method
- Can we define a method inside another method?
- Types of methods
  - Static and non-static
  - Void and non-void
  - Parameterized and Non-Parameterized
  - Final
  - Abstract
  - Native
  - Synchronized
  - Strictfp
- What are the modifiers allowed for a method?
  - CE: modifier not allowed here
- Rules in calling static and non-static methods
  - CE: non-static method cannot be referenced from static context
- Rules in calling parameterized and non-parameterized methods
  - CE: cannot find symbol
- Rules in calling void and non-void methods
  - CE: cannot return a value whose result type is void
  - CE: missing return statement
  - CE: missing return value
  - CE: incompatible types
  - CE: possible loss of precision
  - CE: void type is not allowed here
- Need of break, continue, return statement
- Rules on above three statements
  - CE: unreachable statement
  - CE: break outside loop or switch
  - CE: continue outside loop

Methods and Types of Methods

Definition
Method is a sub block of a class that is used for implementing logic of an object's operation.

Rule: logic must be placed only inside a method, not directly at class level.
If we place logic at class level compiler throws error.

- So at class level we are allowed to place only variable and method creation statements.
- The logical statements such as method calls, validations, calculations, and data printing related statements must be placed inside method, because these statements are considered as logic.

Find out CEs from the below program

```
//Example.java
class Example{
        static int a = 10;
        static int b = a + 10;

        a = 20;
        System.out.println(a + " ... " + b);

        m1();

        if (true){
                System.out.println("Hi");
        }

        public static void main(String[] args){
                System.out.println(a + " ... " + b);

                m1();

                if (true){
                        System.out.println("Hi");
                }
        }
        static void m1(){
                System.out.println("m1");
        }
}
```

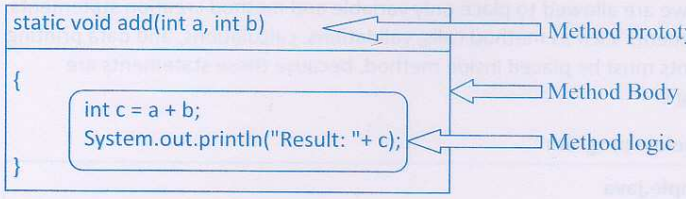
class Level

Method Level

Method Terminology

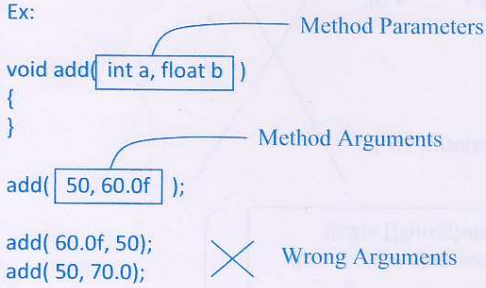
1. **Method prototype**: The head portion of the method is called method prototype.
2. **Method body and logic**: The "{ }" region is called method body, and the statements placed inside method body is called logic.

   Ex:



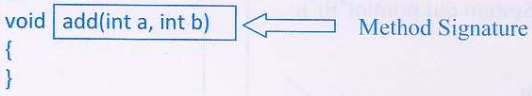
3. **Method parameters and arguments:**
   - The *variables declared* in method parenthesis "( )" are called parameters. We can define method with ZERO to 'n' number of parameters.
   - The *input values passing* to parameters are called arguments. In method invocation we must pass arguments according to method parameters order and type.

   Ex:



4. **Method signature**: The combination of [Method name + parameters list] is called method signature.

   Ex:



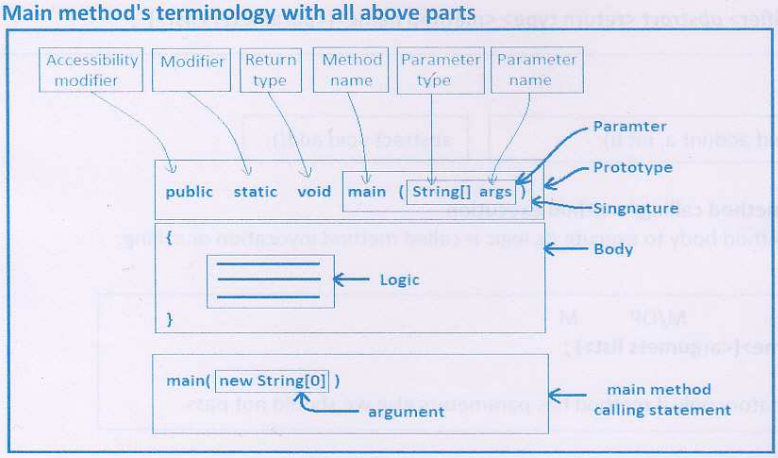
5. **Method return type:**
   - The datatype keyword that is placed before method name is called method return type. It tells to Compiler, JVM and developer about the type of the value is returned from this method after its execution. If we don't want to return value for a method we must use *void* keyword as return type for that method.

***void* return type keyword**

- If we do not want to return any value from a method, we must use "void" as return type. It tells that the method does not return any value.
- If we want to return some value, we must place datatype keyword as return type. For example if we want to return integer value we must place either 'int' or 'long'.

Note: *byte, short* are not allowed because by default integer value type is "*int*"

Main method's terminology with all above parts



Method creation and execution syntax

Defining, declaring and invoking/calling a method:

We can create method in two ways

1. Method creation with body - The process of creating method with body is called method definition. Technically this method is called *concrete* method.

Syntax:

```
        OP          OP        M          M          OP
<Accessibility Modifier> <Modifier> <return type> <method name>(<parameters list>)
{
M                                                          OP
}; OP
```

For Example:

```
public static void add(int a , int b){
        System.out.println( a + b );
};
```

```
void add(){

}
```

2. Creating a method without body - Creating a method without body is called declaring a method / method declaration. Technically this method is called abstract method.
Rule: In method declaration the modifier "***abstract***" is mandatory and also should be terminated with '**;**'.

Syntax:

```
        OP                    M       M                    M          OP          M
<Accessibility Modifier> abstract <return type> <method name>(<parameters list>) ;
```

For example::

```
public abstract void add(int a, int b);
```

```
abstract void add();
```

Method invocation/ method calling/ method execution
Sending cursor to a method body to execute its logic is called method invocation or calling.
Syntax:

```
            M          M/OP      M
    <methodName>(<argumets list>) ;

arguments are mandatory only if method has parameters else we should not pass.
```

Below program explains defining methods and calling them from main method.

```
class Example
{
        static void m1(int a)
        {
                System.out.println("m1");
        }

        static void m2()
        {
                System.out.println("m2");
        }
```

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

                m1(50);

                m2();
        }

}
```

Conclusions from the above program

1. When we call a method control send to that method.
2. If we pass argument that value is stored in parameter variable.
3. After method execution that parameter variable is destroyed and control is sent back to calling method.
4. Control is sent back to calling method with value if method return type is not void.

Types of methods

Basically concrete methods are divided into 3 types

1. Based on *static* modifier we have two types of methods
   a. static methods
   b. non-static methods
2. Based on *return type* we have two types of methods
   a. void methods
   b. non-void methods
3. Based on *parameter* we have two types of methods
   a. parameterized methods
   b. non-parameterized methods.

Static and Non-static methods

If a method has static keyword in its definition (prototype) then that method is called static method, else it is called non-static method.

Ex:

//static method static void m1() { }	**//non-static method** void m1() { }

Calling static and non-static methods

We can call static methods directly from main method, but we cannot call non-static methods directly from main method. It leads CE: "***non-static method cannot be referenced from static context***", because class level members will not get memory directly. JVM provides memory only if we use either "static or new" keywords.

Below program shows calling static and non-static methods

//Example.java

```
class Example {

    static void m1(){
        System.out.println("In m1");
    }

    void m2() {
        System.out.println("In m2");
    }
```

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

        System.out.println("In main");

        m1();
        //m2(); CE: non-static method m2 cannot be referenced from static context.

        //Below we are using new keyword to provide memory for m2() method.
        Example e = new Example();

        //It gets memory with reference to "e" variable.
        //So we must call it as shown below.
        e.m2();
    }
}
```

Void and Non-void methods

If the method return type is 'void', it is called void method; else it is called non-void method.

Rule: Non-void method must return a value after its execution. Also that value type must be compatible with method return type and its range must be less than or equals to method return type. Else it leads to compile time error.

For example:

//**static void** method static void m1() { }	//**non-static void** method void m2() { }
//**static non-void** method static int m1() { return 10; }	//**non-static non-void** method double m2() { return 23.45; }

Q) Are statements optional or mandatory in a method?

In void methods statements are optional, but in non-void methods return statement is mandatory with value range **less than or equals to** method return type range.

Rule: If we do not place return statement in non-void methods compiler throws **CE:**"*missing return statement*"

Types of return statements:

We have two types of return statements

1. return;
2. return <value>;

Rule on return statements

- "*return;*" is only allowed in void methods and constructor, and it is optional.
- "*return <value>;*" is only allowed in non-void methods, and it is mandatory.

Find out compile time errors in below method definitions.

```
1. void m1(){}

2. void m1(){return; }

3. void m1(){
     return 10;
   }

4. int m1(){}

5. int m1(){ return;}

6. int m1(){ return 10;}
7. int m1(){ return 'a';}
8. int m1(){ return 10.345;}

9. int m1(){ return true;}
```

Calling Void and Non-void methods

In general, methods are called in three ways

1. Directly -> m1();
2. As variable initialization statement -> int x = m1();
3. As Sopln() argument -> Sopln(m1());

Non-void method can be called in all three ways.

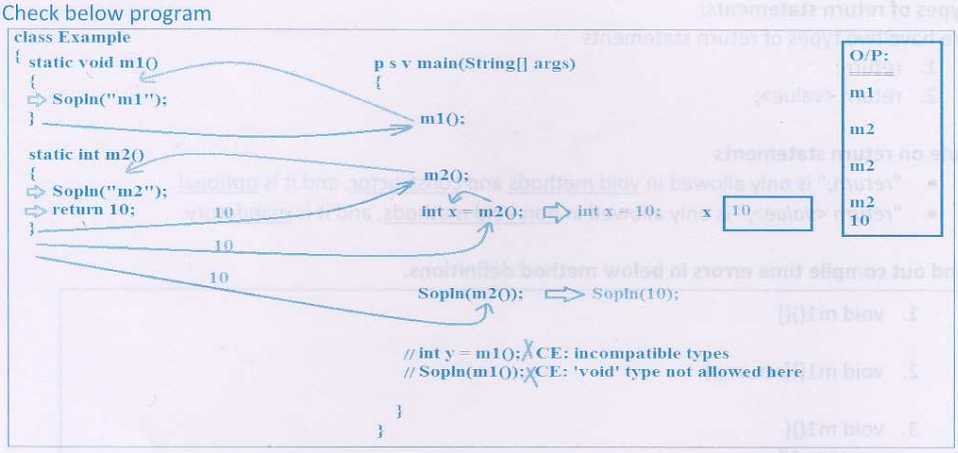
- In the first way the returned value is lost
- In the second way the returned value is stored in the destination variable
- In the third way the returned value is printed on console

Only in second way we can use returned value in further program logic.

Void method can only be called in first way.

If we call it either in second or third ways it leads to CE.

Check below program



Purpose of return statement

Basically return statement is used to terminate method execution and for sending control back to calling method.

- "return;" sends control back to calling method without value.
- "return <value>;" sends control back to calling method with value.

What is the output from the below program?

```
class Example{
        static void m1(int a){
                System.out.println("Before if");
                if(a == 10){
                        System.out.println("In if");
                        return;
                }
                System.out.println("after if");
                System.out.println("Hi");
        }
        static int m2(int a){
        System.out.println("Before if");
                if(a == 10){
                        System.out.println("In if");
                        return a + 10;
                }
                System.out.println("After if");
                System.out.println("Hi");
                return 50;
        }
```

```
public static void main(String[] args){
    m1(10);
    m1(20);

    m2(10);
    m2(20);
  }
}
```

3. Parameterized and Non-parameterized methods

If a method is created with parameters, it is called parameterized method, else it is called non-parameterized | no-arg method | ZERO arg method.

Ex:

```
//non-parameterized
static void m1()
{
}
```

```
//parameterized
static void m1(int x)
{
}
```

Calling parameterized and non-parameterized methods

Parameterized methods must be called by passing the parameter type argument, else it leads to CE: cannot find symbol.

Check below program

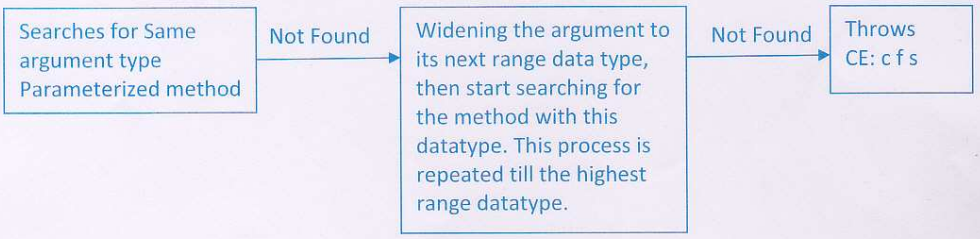
```
class Example
{
        static void m1()
        {
                System.out.println("m1");
        }

        static void m2(int a)
        {
                System.out.println("m2");
        }
```

```
        public static void main(String[] args){
                m1();
                //m1(50); CE: cannot find symbol
                //m2(); CE: cannot find symbol
                m2(50);
                m2('a');
                //m1(50.34); CE: cannot find symbol
                //m1(true);    CE: cannot find symbol
        }
}
```

Conclusion: Searching for Parameterized method definition

Compiler searches for parameterized method definition as shown below



The above flow chart is common for both *primitive* and *referenced datatypes*.
For primitive data types the highest range datatype is *double,* and for referenced data type the highest range datatype is *java.lang.Object,* it is the super class of all referenced datatypes.

//Passing object as an argument and return type –What is the output from below program?

```
class A{}

class Example{

        static void m1(A a){
                System.out.println("m1");
        }

        static A m2(String s){
                System.out.println("m2");
                return new A();
        }

        public static void main(String[] args){
                A a1 = new A();
                m1(a1);

                m2("Hari");
                A a2 = m2("Krishna");

                System.out.println( m2("NareshIT") );
        }
}
```

Note:

- If we pass primitive values as an argument, the value is passed directly and is stored in parameter variable.
- If we pass object as an argument, it's reference is passed not its memory, and that reference is stored in parameter variable. Then that parameter variable is also pointing to the same object as shown above.

Q) What are the modifiers not allowed for a method?

Except *transient* and *volatile* all other 9 modifiers are allowed.