Java

Agenda

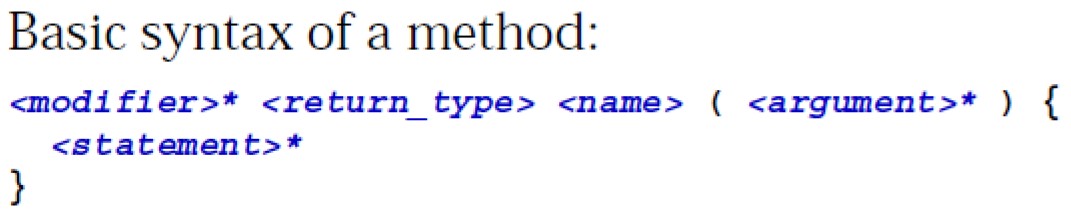
- Method
- Constructor

Method

METHOD????

- A method is a code block that contains a series of statements
- A program causes the statements to be executed by calling the method and specifying any required method arguments
- Every executed instruction is performed in the context of a method.
- The Main method is the entry point for every java application and it is called by the JVM when the program is started.

Method Signature



Basic syntax of a method:

```
<modifier>* <return_type> <name> ( <argument>* ) {
  <statement>*
}
```

CREATE METHOD

- Creating a method :
  - public int funcName(int a, int b) { // body }
    - **public** : Access specifier
    - **int**: return type
    - **funcName**: function name
    - **int a, int b**: list of parameters

Method / Local Variable

Local variables are:

- Variables that are defined inside a method and are called *local*, *automatic*, *temporary*, or *stack* variables
- Variables that are created when the method is executed are destroyed when the method is exited

Variable initialization comprises the following:

- Local variables require explicit initialization.
- Instance variables are initialized automatically.

Method Access

- Calling a method on an object is like accessing a field.
- After the object name, add a period, the name of the method, and parentheses.
- Arguments are listed within the parentheses, and are separated by commas.

  - The *dot notation is:* ***<object>.<member>***

Method Parameters Vs Arguments

- The **method definition** holds the names and types of any **parameters** that are required.
- **Calling code** calls the method, it provides concrete values called **arguments** for each parameter.

- The arguments must be compatible with the parameter type
- The argument name (if any) used in the calling code does not have to be the same as the parameter named defined in the method.

```
/** the snippet returns the minimum between two numbers */
Public Int minFunction(int n1, int n2) {
   int min;
   if (n1 > n2)
      min = n2;
   else
      min = n1;

   return min;
}
```

```
public class ExampleMinNumber
{

   public static void main(String[] args)
{
      int a = 11;
      int b = 6;
      ExampleMinNumber e=new ExampleMinNumber();
      int c = e.minFunction(a, b);
      System.out.println("Minimum Value = " + c);
}

   /** returns the minimum of two numbers */
   public int minFunction(int n1, int n2) {
      int min;
      if (n1 > n2)
         min = n2;
      else
         min = n1;

      return min;
   }
}
```

Pass by reference Vs Pass by value

- When a value type is passed to a method, a copy is passed instead of the object itself.
- Can pass a value-type by reference by using the ref keyword.

- When an object of a reference type is passed to a method, a reference to the object is passed.
- The method receives not the object itself but an argument that indicates the location of the object.

Overloading

Method Overloading

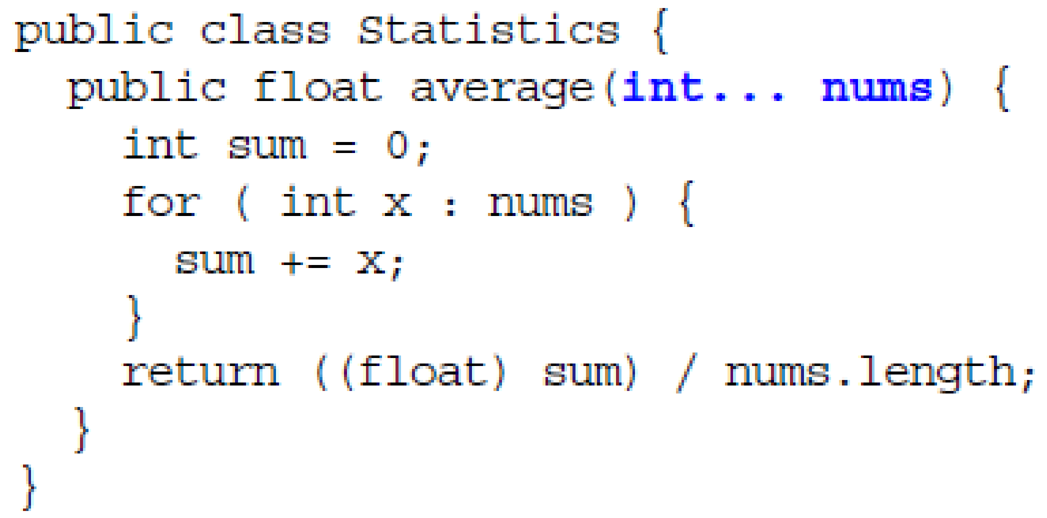
Overloading Methods

- Use overloading as follows:

```
public void println(int i)
public void println(float f)
public void println(String s)
```

- Argument lists *must* differ.
- Return types *can* be different.

Variable Argument Methods



```
public class Statistics {
  public float average(int... nums) {
    int sum = 0;
    for ( int x : nums ) {
      sum += x;
    }
    return ((float) sum) / nums.length;
  }
}
```

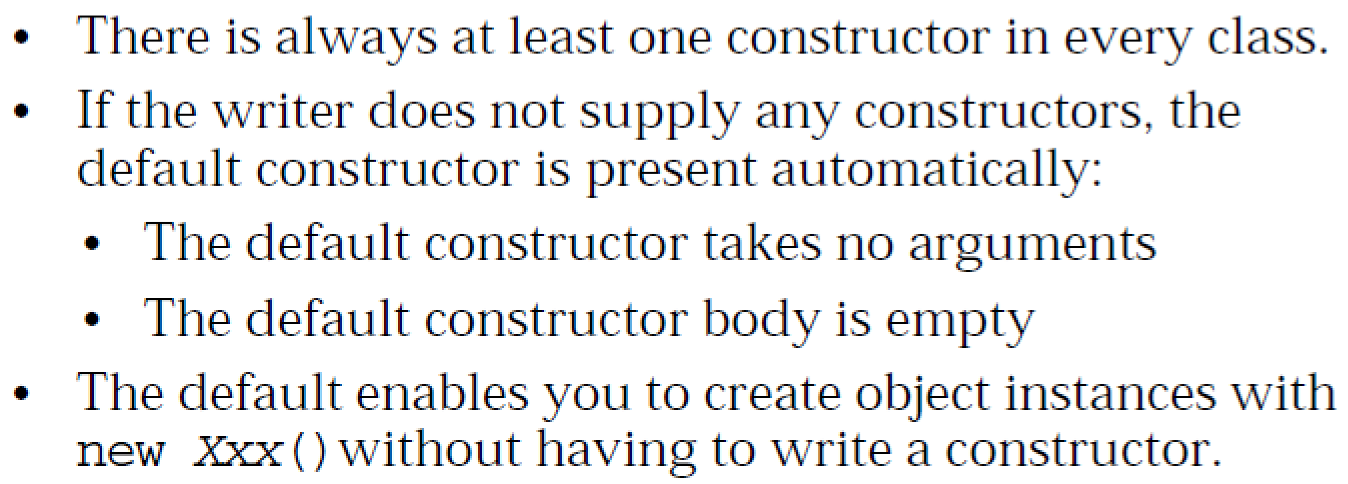
```java
public class VarargsDemo {

   public static void main(String args[]) {
      // Call method with variable args
           printMax(34, 3, 3, 2, 56.5);
      printMax(new double[]{1, 2, 3});
   }

   public static void printMax( double... numbers) {
   if (numbers.length == 0) {
      System.out.println("No argument passed");
      return;
   }

   double result = numbers[0];

   for (int i = 1; i <  numbers.length; i++)
      if (numbers[i] >  result)
      result = numbers[i];
      System.out.println("The max value is " + result);
   }
}
```

Constructor

The Constructor

- A constructor initializes an object when it is created.
- Has the same name as its class
- Have no explicit return type.
- Used to give initial values to the instance variables defined by the class
- All classes have a default constructor that initializes all member variables to zero.
- When own constructor is defined by the programmer, the default constructor is no longer used.

DEFAULT CONSTRUCTOR



- There is always at least one constructor in every class.
- If the writer does not supply any constructors, the default constructor is present automatically:
  - The default constructor takes no arguments
  - The default constructor body is empty
- The default enables you to create object instances with `new` *`Xxx`*`()` without having to write a constructor.

CONSTRUCTOR OVERLOADING

- As with methods, constructors can be overloaded. An example is:

```
public Employee(String name, double salary, Date DoB)
public Employee(String name, double salary)
public Employee(String name, Date DoB)
```

- Argument lists *must* differ.
- You can use the `this` reference at the first line of a constructor to call another constructor.

```
public class Employee {
  private static final double BASE_SALARY = 15000.00;
  private String name;
  private double salary;
  private Date   birthDate;

  public Employee(String name, double salary, Date DoB) {
    this.name = name;
    this.salary = salary;
    this.birthDate = DoB;
  }
  public Employee(String name, double salary) {
    this(name, salary, null);
  }
  public Employee(String name, Date DoB) {
    this(name, BASE_SALARY, DoB);
  }
  // more Employee code...
}
```