Methods

Sayed Ahmad Sahim

Benawa University

September 27, 2017

Road Map

1. Methods

Methods

- A method is a piece of code that is called by a name that is associated with an object. In most respects it is identical to a function except for two key differences:
    1. A method is implicitly passed the object on which it was called.
    2. A method is able to operate on data that is contained within the class (remembering that an object is an instance of a class - the class is the definition, the object is an instance of that data).

Why method

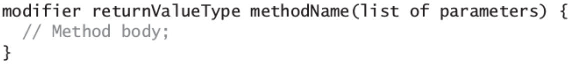
- Method plays an important role in the conceptual design of a program.
- Any sequence of instructions that appears in a program more than once is a candidate for being made into a function which reduces the program size.
- Function code is stored once in memory and can be executed many times in the program.

Access Modifiers in java

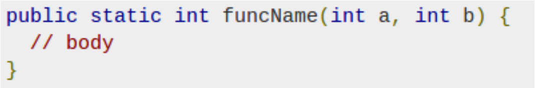
- Java provides a number of access modifiers to set access levels for classes, variables, methods, and constructors.
- there are four access modifiers:
  1. Visible to the package, the default. No modifiers are needed.
  2. Visible to the class only (private).
  3. Visible to the world (public).
  4. Visible to the package and all subclasses (protected).

Declaring a Method

- Syntax:



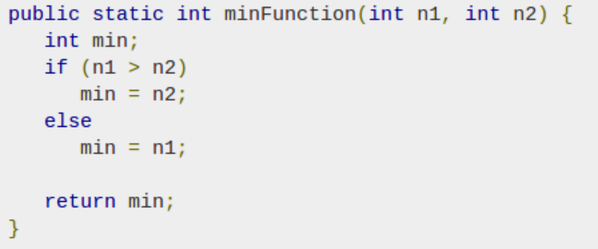
```
modifier returnValueType methodName(list of parameters) {
  // Method body;
}
```



```
public static int funcName(int a, int b) {
  // body
}
```

1. public static : modifier
2. int: return type
3. funcName: function name
4. a, b: formal parameters
5. int a, int b: list of parameters

Example

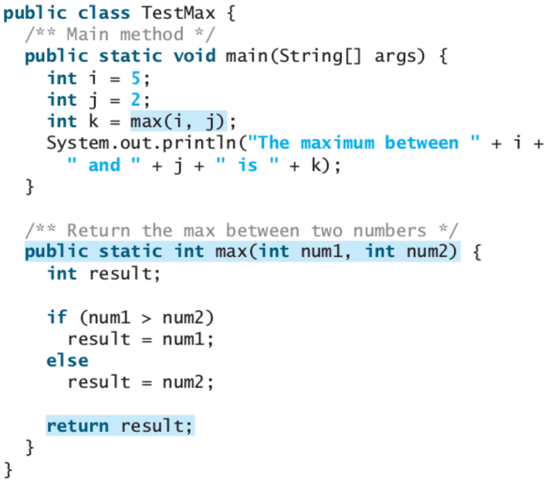


```
public static int minFunction(int n1, int n2) {
   int min;
   if (n1 > n2)
      min = n2;
   else
      min = n1;

   return min;
}
```

Calling a Method

- Declaring a method, you define what the method is suppose to do.
- To use a method, you have to call or invoke it.
- There are two ways to call a method, depending on whether the method returns a value or not:
  1. If the method returns a value, a call to the method is usually treated as a value. For example *int larger = max(3, 4);* or *System.out.println(max(3, 4));*
  2. If the method returns void , a call to the method must be treated as a statement. *System.out.println("Welcome to Java!");*

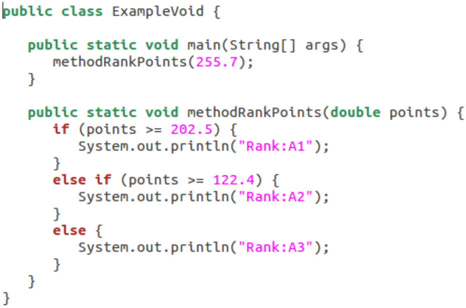
Example



```
public class TestMax {
  /** Main method */
  public static void main(String[] args) {
    int i = 5;
    int j = 2;
    int k = max(i, j);
    System.out.println("The maximum between " + i +
      " and " + j + " is " + k);
  }

  /** Return the max between two numbers */
  public static int max(int num1, int num2) {
    int result;

    if (num1 > num2)
      result = num1;
    else
      result = num2;

    return result;
  }
}
```

The void Keyword

The void keyword allows us to create methods which do not return a value. A call to a void method must be a statement.

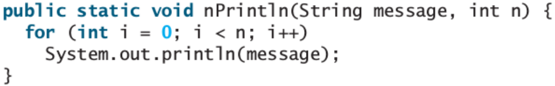


```
public class ExampleVoid {

   public static void main(String[] args) {
      methodRankPoints(255.7);
   }

   public static void methodRankPoints(double points) {
      if (points >= 202.5) {
         System.out.println("Rank:A1");
      }
      else if (points >= 122.4) {
         System.out.println("Rank:A2");
      }
      else {
         System.out.println("Rank:A3");
      }
   }
}
```

Passing Parameters by Values

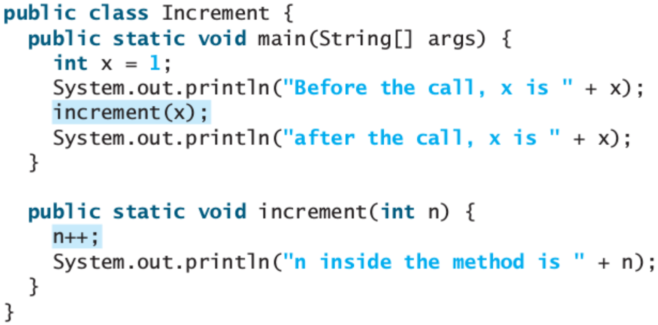
- Parameters are the variables provided to methods as an input data.
- While calling the method variables values should be provided in the same order as they are defined in method declaration.

For example, the following method prints a message n times:



```
public static void nPrintln(String message, int n) {
  for (int i = 0; i < n; i++)
    System.out.println(message);
}
```

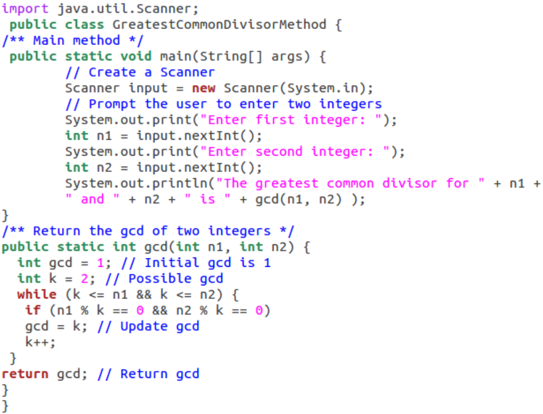
Pass by value example



```java
public class Increment {
  public static void main(String[] args) {
    int x = 1;
    System.out.println("Before the call, x is " + x);
    increment(x);
    System.out.println("after the call, x is " + x);
  }

  public static void increment(int n) {
    n++;
    System.out.println("n inside the method is " + n);
  }
}
```

Modularizing Code

- Methods can be used to reduce redundant code and enable code reuse.
- Methods can also be used to modularize code and improve the quality of the program.



```
import java.util.Scanner;
 public class GreatestCommonDivisorMethod {
/** Main method */
 public static void main(String[] args) {
        // Create a Scanner
        Scanner input = new Scanner(System.in);
        // Prompt the user to enter two integers
        System.out.print("Enter first integer: ");
        int n1 = input.nextInt();
        System.out.print("Enter second integer: ");
        int n2 = input.nextInt();
        System.out.println("The greatest common divisor for " + n1 +
        " and " + n2 + " is " + gcd(n1, n2) );
}
/** Return the gcd of two integers */
public static int gcd(int n1, int n2) {
  int gcd = 1; // Initial gcd is 1
  int k = 2; // Possible gcd
  while (k <= n1 && k <= n2) {
   if (n1 % k == 0 && n2 % k == 0)
   gcd = k; // Update gcd
   k++;
 }
return gcd; // Return gcd
}
}
```

Advantages of using modularization

- By encapsulating the code for obtaining the gcd in the previous method can have several advantages:
  1. It isolates the problem for computing the gcd from the rest of the code in the main method. Thus, the logic becomes clear and the program is easier to read.
  2. The errors on computing gcd are confined in the gcd method, which narrows the scope of debugging.
  3. The gcd method now can be reused by other programs.

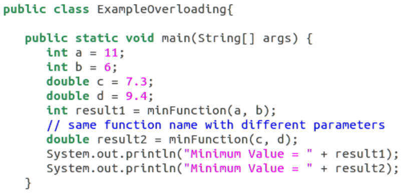
Overloading Methods

- When a class has two or more methods by same name but different parameters, is known as method overloading.
- Method overloading is different from method overriding.
- In overriding a method has same method name, type, number of parameters etc.

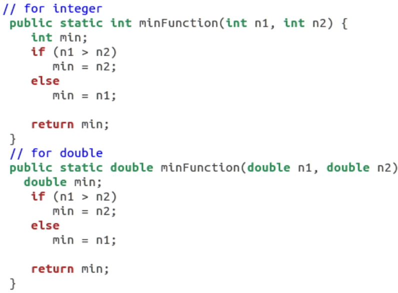
Method overloading

Example

Lets consider the example shown before for finding minimum integer number. lets say we want to find minimum number of double type.



```
public class ExampleOverloading{

   public static void main(String[] args) {
      int a = 11;
      int b = 6;
      double c = 7.3;
      double d = 9.4;
      int result1 = minFunction(a, b);
      // same function name with different parameters
      double result2 = minFunction(c, d);
      System.out.println("Minimum Value = " + result1);
      System.out.println("Minimum Value = " + result2);
   }
```



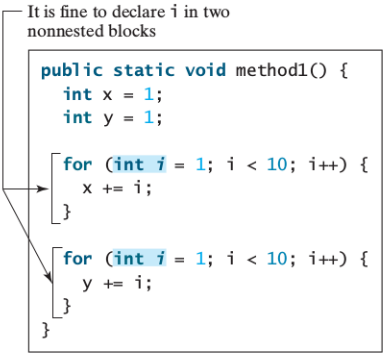
```
// for integer
 public static int minFunction(int n1, int n2) {
    int min;
    if (n1 > n2)
       min = n2;
    else
       min = n1;

    return min;
 }
 // for double
 public static double minFunction(double n1, double n2)
   double min;
    if (n1 > n2)
       min = n2;
    else
       min = n1;

    return min;
 }
```

Variables Scope

- The scope of a variable defines the section of the code in which the variable is visible.
- As a general rule, variables that are defined within a block are not accessible outside that block.
- The lifetime of a variable refers to how long the variable exists before it is destroyed.

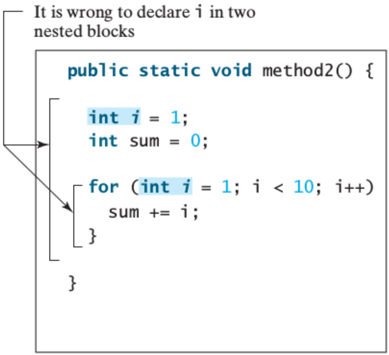
- **Instance variables** Instance variables are those that are defined within a class itself and not in any method or constructor of the class.
- **Argument variables** These are the variables that are defined in the header of a constructor or a method.
- **Local variables** A local variable is the one that is declared within a method or a constructor (not in the header). The scope and lifetime are limited to the method itself.

Variable Scope



It is fine to declare i in two nonnested blocks

```
public static void method1() {
  int x = 1;
  int y = 1;

  for (int i = 1; i < 10; i++) {
    x += i;
  }

  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```



It is wrong to declare i in two nested blocks

```
  public static void method2() {

    int i = 1;
    int sum = 0;

    for (int i = 1; i < 10; i++)
      sum += i;
    }

  }
```

Java built-in functions

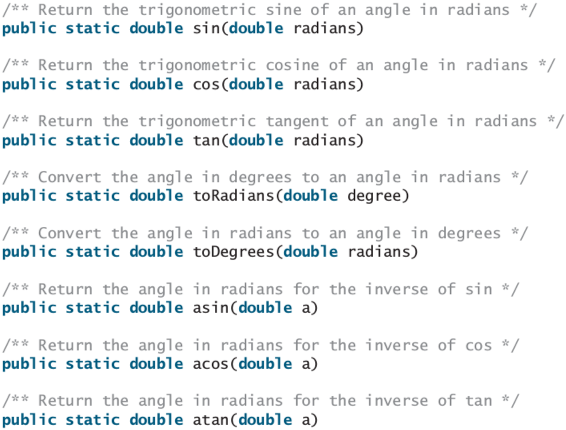
- Built in functions in java are methods that are present in different API of JDK.
- For example cos(double a), exp(double a) etc are built in function of java present in java.lang.Math class.

The Math Class

The Math class contains the methods needed to perform basic mathematical functions. You have already used the pow(a, b) method to compute a^b

Trigonometric Methods

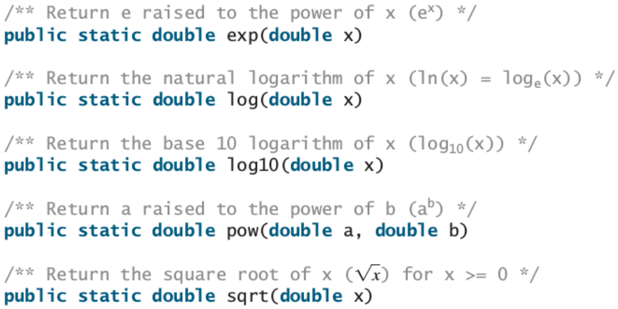
The Math class contains the following trigonometric methods



```
/** Return the trigonometric sine of an angle in radians */
public static double sin(double radians)

/** Return the trigonometric cosine of an angle in radians */
public static double cos(double radians)

/** Return the trigonometric tangent of an angle in radians */
public static double tan(double radians)

/** Convert the angle in degrees to an angle in radians */
public static double toRadians(double degree)

/** Convert the angle in radians to an angle in degrees */
public static double toDegrees(double radians)

/** Return the angle in radians for the inverse of sin */
public static double asin(double a)

/** Return the angle in radians for the inverse of cos */
public static double acos(double a)

/** Return the angle in radians for the inverse of tan */
public static double atan(double a)
```

Exponent Methods

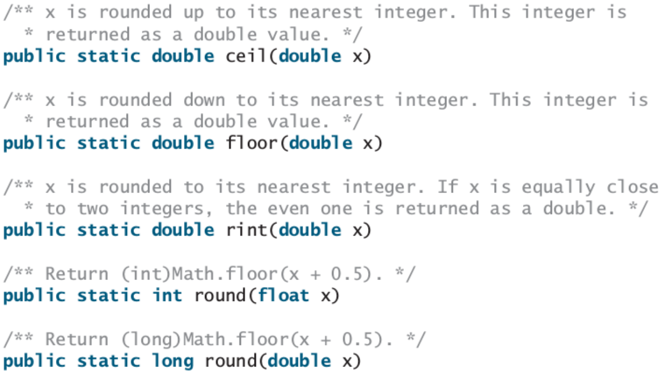
There are five methods related to exponents in the Math class:



```
/** Return e raised to the power of x (e^x) */
public static double exp(double x)

/** Return the natural logarithm of x (ln(x) = log_e(x)) */
public static double log(double x)

/** Return the base 10 logarithm of x (log_10(x)) */
public static double log10(double x)

/** Return a raised to the power of b (a^b) */
public static double pow(double a, double b)

/** Return the square root of x (√x) for x >= 0 */
public static double sqrt(double x)
```

The Rounding Methods

The Math class contains five rounding methods:



```
/** x is rounded up to its nearest integer. This integer is
  * returned as a double value. */
public static double ceil(double x)

/** x is rounded down to its nearest integer. This integer is
  * returned as a double value. */
public static double floor(double x)

/** x is rounded to its nearest integer. If x is equally close
  * to two integers, the even one is returned as a double. */
public static double rint(double x)

/** Return (int)Math.floor(x + 0.5). */
public static int round(float x)

/** Return (long)Math.floor(x + 0.5). */
public static long round(double x)
```