Methods

Methods

- It is a program module that contains a series of statements that carry out a task.
- main() method - which executes automatically when you run a program
- Any class can contain an unlimited number of methods, and each method can be called an unlimited number of times.
- To execute a method, you invoke or call it.

Example

```
public class First {
   public static void main(String[] args){
      System.out.println("XYZ Company");
      System.out.println("8900 U.S. Hwy 14");
      System.out.println("Crystal Lake, IL 60014");
      System.out.println("First Java application");}
}
```

Example

```
public class First {
    public static void main(String[] args){
        nameAndAddress();
        System.out.println("First Java application");
    }
    public static void nameAndAddress(){
        System.out.println("XYZ Company");
        System.out.println("8900 U.S. Hwy 14");
        System.out.println("Crystal Lake, IL 60014");
    }
}
```

Methods

- Are written and invoked or called
- Are reusable
- Are placed within a class
- Can never overlap
  - it must be outside of any other methods.

```
public class First{
    //You can place additional methods here, before main()
    public static void main(String[] args){
        nameAndAddress();
        System.out.println("First Java application");
    }
    // You can place additional methods here, after main()
}
```

Methods

- The order of methods in a class has no bearing on the order in which the methods are called or executed.
- A main() method executes automatically when you run a program
- But other methods do not execute simply because you place them within a class—they must be called.
- abstraction

Method Construction

- method header
  - provides information about how other methods can interact with it
  - called as declaration
- Method body
  - contains the statements that carry out the work of the method
  - called as implementation

Method header/ declaration

```
public class First {
    public static void main(String[] args){
        nameAndAddress();
        System.out.println("First Java application");
    }
    public static void nameAndAddress(){
        System.out.println("XYZ Company");
        System.out.println("8900 U.S. Hwy 14");
        System.out.println("Crystal Lake, IL 60014");
    }
}
```

Method body/ implementation

```
public class First {
    public static void main(String[] args){
        nameAndAddress();
        System.out.println("First Java application");
    }
    public static void nameAndAddress(){
        System.out.println("XYZ Company");
        System.out.println("8900 U.S. Hwy 14");
        System.out.println("Crystal Lake, IL 60014");
    }
}
```

Parts of a method header

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

- Optional access specifiers/ access modifier
  - public, private, protected
    - public – any other class can use it
    - private – within the class only
    - protected – superclass subclass relationship only
  - static modifier means that these methods do not require an object to be created

Parts of a method header

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

- Return type
  - void, int, double, string, Boolean
  - describes the type of data the method sends back to its calling method
  - a method that returns no data has a return type of void

Parts of a method header

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

- Method Name/ Identifier
  - Any legal identifier
  - must be one word with no embedded spaces, and cannot be a Java keyword

Parts of a method header

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

- Parentheses
  - always placed just after the method name
  - may or may not contain a data/ parameter
  - If with parameter
    - data type followed by any valid identifier

Adding parameters to methods

- Data items you use in a call to a method are called arguments.
- When the method receives the data items, they are called parameters.

- Single Parameters

```
public static void main (String[] args){
    int x = 5;
    square(x);}
public static void square(int y){
    int z = y * y;
    System.out.println(z);}
```

- Multiple Parameters

```
public static void main (String[] args){
    int x = 5, y=10;
    calculate(x);}
public static void calculate(int a, int b){
    int z = a + b;
    System.out.println(z);}
```

```
public static void main(String[] args){
   double x = 200.00;
   double y = 800.00;
   System.out.println("Demonstrating");
   predictRaise(400.00);
   predictRaise(x);
   predictRaise(y);
}

public static void predictRaise(double x){
   double z;
   final double a = 1.10;
   z = x * a;
   System.out.println(x + "-" + z);
}
```

Creating Methods That Return Values

- return type for a method can be any data type used in Java as well as class types
- return statement causes a method to end and the program's logic to return to the calling method
- return statement frequently sends a value back to the calling method
- the type of the value used in the return statement must match the method's declared return type
- the returned value can be a named or unnamed constant, as long as it is the correct data type or can be automatically converted to the correct data type.
- cannot place any statements after a method's return statement (unreachable statements/dead code)

```
public static double predictRaise(double salary) {
    double newAmount;
    final double RAISE = 1.10;
    newAmount = salary * RAISE;
    return newAmount;
}
```

```
//Chaining Method Calls
public static double predictRaise(double salary){
    double newAmount;
    double bonusAmount;
    final double RAISE = 1.10;
    newAmount = salary * RAISE;
    bonusAmount = calculateBonus(newAmount);
    newAmount = newAmount + bonusAmount;
    return newAmount;
}

public static double calculateBonus(double salary){
    final double BONUS_AMT = 50.00;
    salary = salary + BONUS_AMT;
    return salary;
}
```