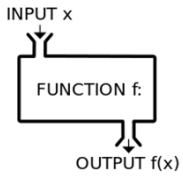
Methods

Also known as Functions

What's a method?

- Methods are Functions
- But there are some differences from what you think of as functions from math



Differences

Math

- Function names are usually letters; e.g. f(x) = x + 3
- Functions have one or more inputs
- Functions have one output
- The type of inputs for the function are not stated
- The type of output the function produces is not stated

Computer Programming

- Method names are descriptive; e.g. setName(x)
- Methods can have zero inputs; e.g. setID()
- Methods perform instructions, may not have an output
- Java methods must have input types specified
- Java functions must have an output type specified

Method Headers

- Method headers should have these 4 things:
  - **Access modifier**: public, private
  - **Return type**: void, int, double, boolean, String
  - **Method name**: e.g. withdraw
  - **Parameter list**: e.g. String pass, double amt
  - (Some methods may also be static)
- public void withdraw(String pass, double amt)

Main Method

The **`main`** method is our first, and so far only, example of a method. When you tell Java (the JVM) to run a compiled class file, it automatically looks for the **`main`** method and runs the code inside the body of the **`main`** method.

So the **`main`** method is our starting point.

But we can have as many other methods as we want.

Think of each method as a named procedure.

Just like variables hold data that we want to store and use again later by name, methods hold blocks of code that we want to store and use again later by name.

Declaring new methods

Methods are declared inside classes, but outside of **`main`**. The code inside a method's block is known as its body:

```
public class CardDealer
{
  public static void main(String[] args)
  {
    // main code goes here
  }
  public static void shuffle()
  {
    // this is the body of shuffle
  }
}
```

Calling methods

Once we declare a method, we can then call it by name. Calling the method tells Java to run the code inside the method's body. Note the parenthesis after the method name:

```
public class CardDealer
{
  public static void main(String[] args)
  {
    shuffle();
  }
  public static void shuffle()
  {
    // this code will run
  }
}
```

Methods are reusable

We can have many methods in a class, and call them as many times as we want.

```
public class CardDealer
{
  public static void main(String[] args)
  {
    shuffle();
    shuffle();
    deal();
    shuffle();
  }
  // the shuffle and deal methods must be
  // defined here
}
```

In the class is in the class

The relative location of method declarations within a class does not matter. In the previous example, **`main`** was declared first. This is a common practice, but not required or universal. Notice that we could call **`shuffle()`** from within **`main`**, even though its definition comes later.

This is because Java scans the entire class declaration before starting to run the **`main`** method. By the time **`main`** is running, the JVM knows where to find the body code for any other methods you call in the class.

Return value

- If your method is not void, it must have a return value
- The type of the return value is specified by the method header
- Your method must have a return statement to give the output of the method

```
public int square(int x) {
    return x*x;
}
```

Declaration vs Invocation

When you declare a method, it has a method header and the code it runs in { }

- public void withdraw(String pass, double amt) { //code to withdraw money }

When you invoke or call a method, you use the method name and any arguments

- withdraw("1234pass", 200.00);

Parameters vs Arguments

Parameters are not the same as arguments

Parameters are the expected inputs for the method

- public void withdraw(String pass, double amt)

Arguments are the actual values that are passed to the method as inputs when you invoke/call the function

- withdraw("1234pass", 200.00);

Print vs Return

return does not produce an output for your system console

System.out.println() does not give a return value for your method

- You should NEVER use System.out.println() in a method that has a return value (except if you need to debug something in your method).

To test your method, call the method and print the value returned from the method

- System.out.println(square(5));

Naming conventions

Method names and variable names share the same naming rules and conventions: The should always start with a lowercase letter. Subsequent words within the name should be capitalized. The name should be descriptive and accurate.

Since methods represent a set of steps or instructions, their names should be verbs or verb phrases.

Good method names	Bad method names
resetTimer() displayMenu() printResult() showAnswer()	x() menu() printer() ShowAnswer()