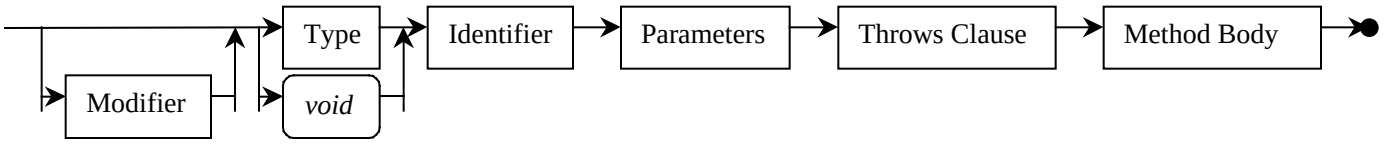
CSCI 1200 Introduction to Computing
More on Writing Classes

Objects & Classes

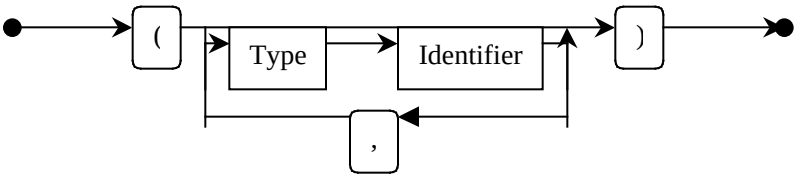
- An object is an instance of a class.
- A class is a blueprint for an object.

Attributes and Methods

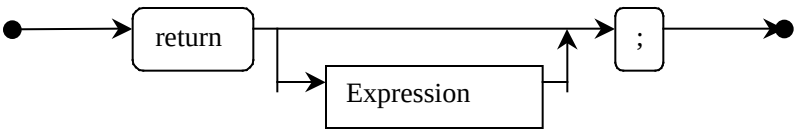
- A method declaration specifies the code that is executed when the method is invoked.
- The syntax of method declaration:



- **Modifiers(optional): final, public, private,…**
  - final: the method/variable returns a constant, and this value may not be changed afterwards.
  - public: the method/variable can be directly referenced from outside of the object.
  - private: the method/variable cannot be referenced externally but still can be used anywhere inside the class definition.
  - Some modifiers can be used together, whereas other combinations are invalid. (ie: a method cannot be private and public, but it can be final and private.)
- **Return Type:** indicates the type of value that will be returned by the method, which may be a primitive type, class name, or the reserved word *void*. A return type declaration of *void* indicates that the method does not return a value.
- **Method name:** an identifier
- **Parameters (optional):**
  - A parameter is a value that is passed into a method when it is invoked.
  - The parameter list in the header of a method specifies the types of the values that are passed and the names by which the called method will refer to the parameters.
  - Formal parameters: the names of the accepted parameters.
  - Actual parameters: the values passed into a method.
  - Syntax of parameter list in the method declaration:



- **Throws clause (optional):** indicates the exceptions that may be thrown by this method.
- **Method Body:** a block of statements that executes when the method is invoked.
  - The *return* statement:
    - A method that returns a value must have a *return* statement. Otherwise, the method usually does not contain a *return* statement. Constructors do not have a *return* statement.
    - Syntax of the *return* statement:

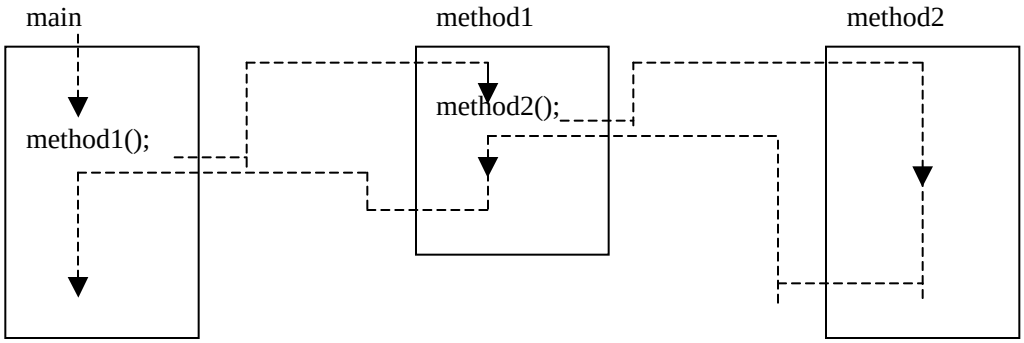


class is instantiated, usually used to initialize the object to a specific initial state. For example, the constructor may assign initial values to some of the variables of the class.

- The name of a constructor is the same as the class name.
- A constructor cannot return a value.
- A class does not have to have a constructor method.

- **Variable Scope/Visibility**: 3 levels:
  - A variable declared in a method is local to that method and cannot be used outside of it. This means that it cannot be used in other methods, even inside the same class.
  - A variable declared in a class and modified as private can be used anywhere inside that class, but not outside of it.
  - A variable declared in a class and modified as public can be used anywhere inside that class and anywhere outside of it.

- **Nested method calls**: In the example below, method1 invokes method2 and the main method invokes method1. Thus, for example, it is possible to pass a value from method2 to the main method via method1.



Example:

```
public class Bank_Account
{
        private double balance;

        public static void main (String[] args) {
            Bank_Account bob_Account;
            Bank_Account kay_Account;

            bob_Account = new Bank_Account(100.00);
            kay_Account = new Bank_Account(500.00);

            bob_Account.debit(50.00);
            kay_Account.credit(100.50);
            System.out.println(bob_Account.current_balance());
            System.out.println(kay_Account.current_balance());
        } //end of main method

        void debit(double amount) {
            this.balance = this.balance - amount;
           return;
        } //end of debit method

        void credit(double amount) {
```

```
        return;
    } //end of credit method

    double current_balance() {
        return(this.balance);
    } //end of current_balance method

    public Bank_Account(double initial_amount) {
        this.balance = initial_amount;
        return;
    } //end of Bank_Account constructor method

} //end of class
```

Parameter passing in methods

- A parameter is a value that is passed into a method when it is invoked.
- **Formal parameters**: the names of the accepted parameters.
- **Actual parameters**: the values passed into a method.
- **Call by value:** the current value of the actual parameter is copied into the formal parameter in the method header. The formal parameter is a separate copy of the value that is passed in, so any changes made to it have **no effect** on the actual parameter. **Call by value when the type of the parameter belongs to primitive types, such as int, float, boolean and so on.**
- **Call by reference:** when the parameter is passed to a method, the value that gets copied is the address of the object. Therefore, the formal parameter and the actual parameter become aliases of each other. Changes to the object through the reference will be reflected in the calling method. **Call by reference when the type of the parameter is class or array**.

The *Static* Modifier

Static Variables

- **Local variable** – variables declared inside a method
- **Instance variable** – variables declared in a class but not inside a method. Each object has distinct memory for each instance variable, so that each object has distinct value for each variable.
- **Static variable** – variable shared among all instances of a class. There is only one copy of a static variable for all object of a class.

  For example: private static int count=0;

· **Static Methods**

- A static method can be invoked through the class itself.
- **Main method** must be static.
- Math class has many static methods, for example

  i= Math.sqrt(2));
- Example page 233, listing 5.4 & 5.5