Advanced Programming Techniques in Java

COSI 12B

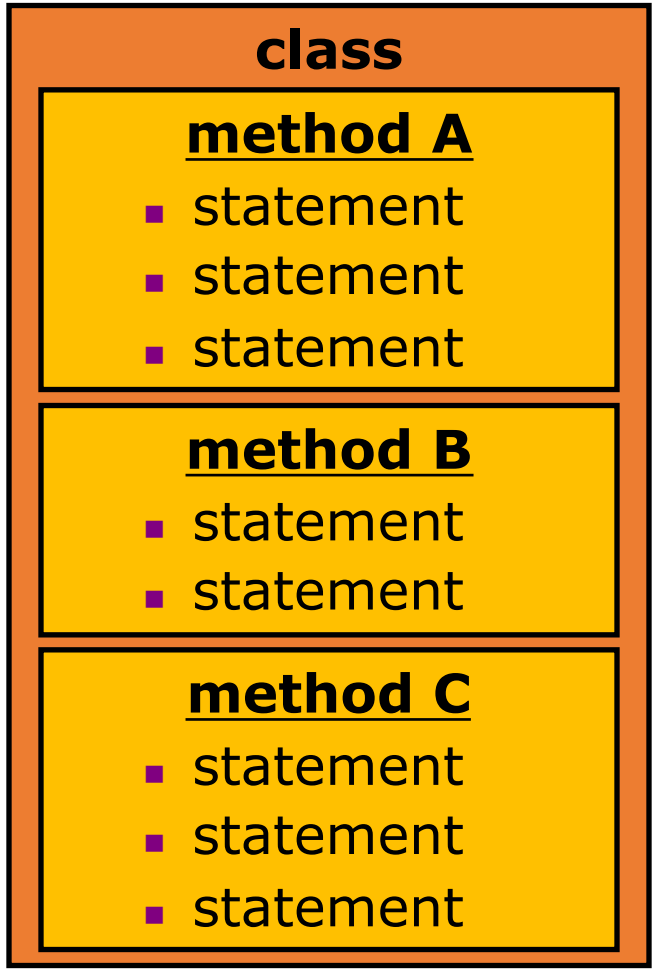
Objectives

- Java Syntax Overview

Review: Java Program Structure

- In the Java programming language:
  - A program is made up of one or more *classes*
  - A class contains one or more *methods*
  - A method contains program *statements*

- A Java application always contains a method called `main`



Review: Object Oriented Design in Java

- Classes and Objects
  - Class definitions in .java files
  - Def: a *class* is a named description for a group of entities that have the same characteristics
  - *Objects* or *instances* of the class is the group of entities
  - The characteristics are the attributes (*data fields*) for each object and the operations (*methods*) that can be performed on these objects

Review: hello!

```
public class Hello{
        public static void main(String[] args){
                System.out.println("Hello World");
        }
}
```

- Everything in Java must be inside a class
- Every file may only contain one public class
- The name of the file must be the name of the class appended to the **`java`** extension

- Thus, **`Hello.java`** must contain one public class named **`Hello`**

Formatting text with `printf()`

- <u>Syntax</u>

```
System.out.printf("format string", <list parameters>);
```

- The *format string* is like placeholders where the parameters are inserted
  - These placeholders are used instead of + concatenation
    - %d integer
    - %f real numbers
    - %s string
- <u>Example</u>

```
int x = 3;
int y = -17;
System.out.printf("x is %d and y is %d\n", x, y);
```

Note: `printf()` does not drop to the next line unless you use `\n`

`printf` precision

`%.Df` real number, rounded to **D** digits after decimal

`%W.Df` real number, **W** characters wide, **D** digits after decimal

```
double gpa = 3.253764;
System.out.printf("your GPA is %.1f\n", gpa);
System.out.printf("more precisely: %8.3f\n", gpa);
```

<u>Output</u>

```
your GPA is 3.3
more precisely:   3.254
```

8

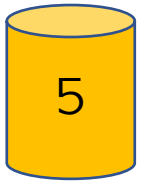
printf with Strings

A simple string	`printf("'%s'", "Hello");`	`'Hello'`
A string with a minimum length	`printf("'%10s'", "Hello");`	`'     Hello'`
Minimum length, left-justified	`printf("'%-10s'", "Hello");`	`'Hello     '`

Variables and Data Types

Variables

- A **variable** is a name for a location in memory
  - It can be thought of as a container which holds values for you



- A variable must be declared by specifying the variable's name and the type of information that it will hold

```
int total;
```

data type

variable name

Variables

- In order to use a variable in a program you to need to perform 2 steps:
  - Variable Declaration
  - Variable Initialization

- A variable can be given an initial value in the declaration

```
int total = 0;
```

Assignment

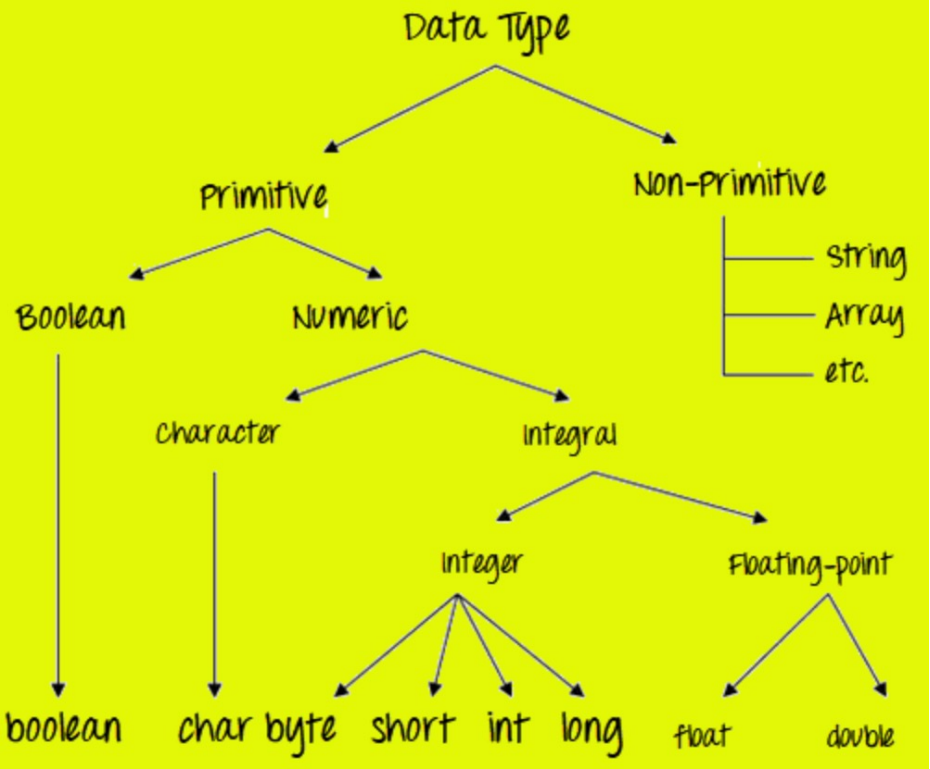
- An *assignment statement* changes the value of a variable

```
total = 35;
```

- The value that was originally in `total` is overwritten
- You can assign only a value to a variable that is consistent with the variable's declared type

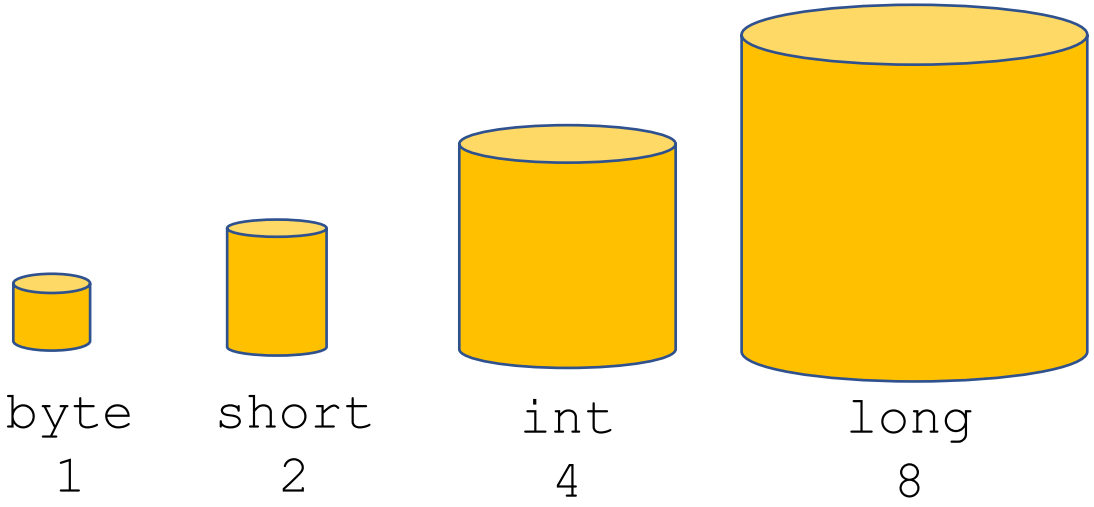
Data Types

- Data types classify the different values to be stored in the variable
- In Java there are two types of data types:
  - Primitive Data Types
  - Non-primitive Data Types



Primitive Data Types

- Primitive Data Types are predefined and available within the Java language
- There are 8 primitive types: `byte, short, int, long, char, float, double, and boolean`



Data type	Default Value	Default size
byte	0	1 byte
short	0	2 bytes
int	0	4 bytes
long		8 bytes
float	0.0f	4 bytes
double	0.0d	8 bytes
boolean	false	1 bit
char	'\u0000'	2 bytes

Primitive Data Types

- byte -128 to 127
- short -32,768 to 32,767
- int -2,147,483,648 to 2,147,483,647
- long -9,223,372,036,854,775,808 to …
- float $\pm 10^{38}$ incl. 0 with 6 digits of precision
- double $\pm 10^{308}$ incl. 0 with 15 digits of precision
- char Unicode character set
- boolean true, false

Example

```
public class ChangeAdder {
        public static void main(String[] args){
                int quarters = 10;
                int dimes = 3;
                int nickels = 7;
                int pennies = 6;
                int change = 0;
                change = 25*quarters+10*dimes+5*nickels+pennies;
                System.out.println("total in change is:" + change);
        }
}
```

Java basics

Data Conversions

- Sometimes it is convenient to convert data from one type to another
- For example, we may want to treat an integer as a floating-point value during a computation
- Conversions must be handled carefully to avoid losing information
- Data conversions can occur in three ways:
  - Assignment conversion
  - Arithmetic promotion
  - Casting

Data Conversions

- *Assignment conversion* occurs when a value of one type is assigned to a variable of another
- *Arithmetic promotion* happens automatically when operators in expressions convert their operands
- *Casting* is accomplished by explicitly casting a value
  - To cast, the type is put in parentheses in front of the value being converted
  - For example, if `total` and `count` are integers, but we want a floating-point result when dividing them, we can cast `total`:

```
result = (double) total / count;
```

Type cast operator

Operators

- Operators are symbols that perform operations on variables and values
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Unary Operators
  - Assignment Operators

Operator	Operation
*, /, %	Multiplication, Division, Mod
+, −	Addition, Subtraction
==	equals
!=	does not equal
<, >	less than, greater than
<=, >=	less than or equal to, greater than or equal to
&&	and
||	or
!	not
++, --, −	increment, decrement, negative
(type)	casting
=, +=, -=, *=, /=, %=	Assignment operators

String Concatenation Operator

- This operator combines several strings into a single string, or combines a string with other data into a new longer string

- Example:

```
System.out.println("Grade: " + (95.1 + 71.9) / 2);
```

- Output:

```
Grade: 83.5
```

Class Libraries and Packages

- A *class library* is a collection of classes that we can use when developing programs
- The `System` class, the `Scanner` class, and the `String` class are part of the Java standard class library
- Related classes are grouped into packages

Package	Purpose
java.lang	General support
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
java.util	Utilities
…	…

Interactive Programs

- The `Scanner` class is used to get input from the user, allowing a program to be interactive
- It is part of the `java.util` package

- A `Scanner` object can read input from many sources:
  - The console window (`System.in`)
  - Files, web sites, databases, …

The `import` Declaration

- In order to access a package, you need to include an `import` declaration
  - You can *import* the class, and then use just the class name, e.g., `import java.util.Scanner;`
  - To import all classes in a particular package, you can use the * wildcard character

    `import java.util.*;`

- All classes of the `java.lang` package (e.g., `System, String, Math`) are imported automatically into all programs

Scanner class

- First a Scanner object is created

```
Scanner <variable-name> = new Scanner(System.in);
```

This parameter tells the constructor that we want the Scanner to read from the standard input (i.e., the keyboard)

- <u>Example</u>: `Scanner console = new Scanner(System.in);`
- Then various methods can be used to read different types of data from the keyboard
- <u>Example</u>: `int num = scan.nextInt();`

`Scanner` methods

Method	Description
`nextInt()`	reads an `int` from the user and returns it
`nextDouble()`	reads a `double` from the user and returns it
`next()`	reads a one-word `String` from the user and returns it
`nextLine()`	reads a one-*line* `String` from the user and returns it

```
Scanner console = new Scanner(System.in);
System.out.print("How old are you? ");  // prompt
int age = console.nextInt();
System.out.println("You typed " + age);
```

Example

```
import java.util.*;
public class UserInputExample {
   public static void main(String[] args) {
      Scanner console = new Scanner(System.in);
      System.out.println("How old are you? ");
      int age = console.nextInt();
      int years = 65 - age;
      System.out.println(years + " years to retirement!");
   }
}
```

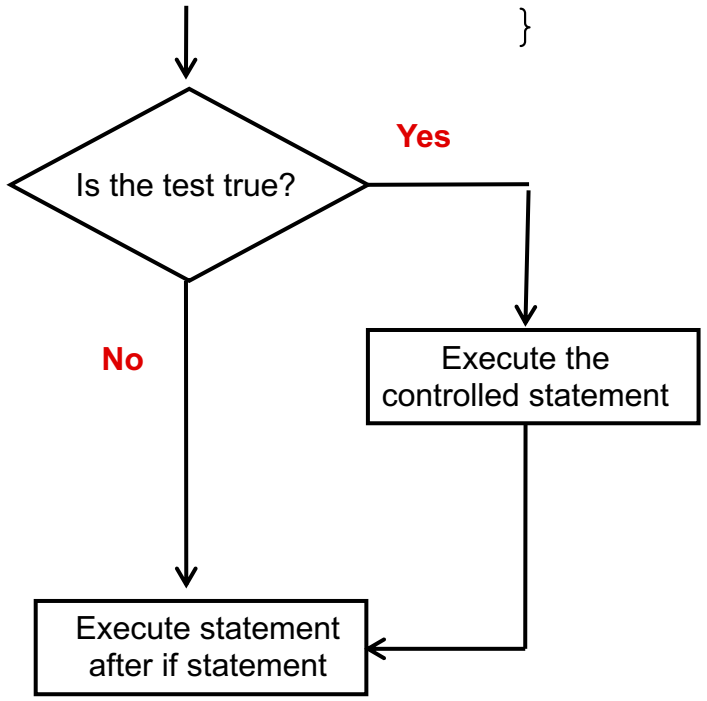
- Console window:

```
How old are you?   29
36 years to retirement!
```

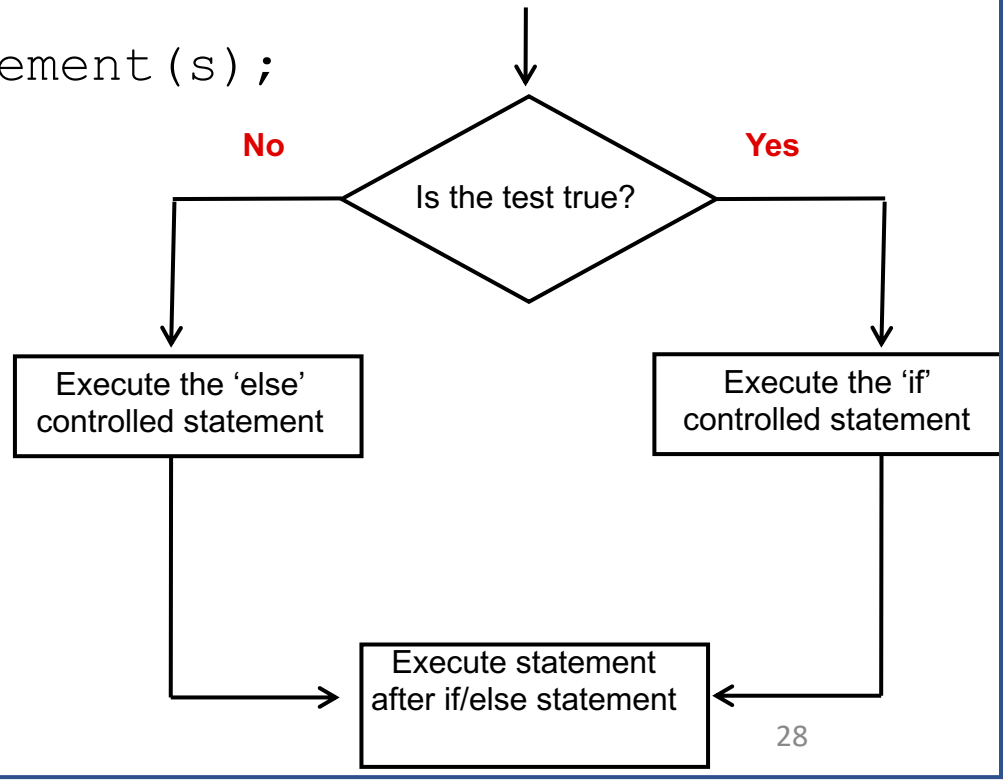
Conditional Statements

- A *conditional statement* lets us choose which statement will be executed next

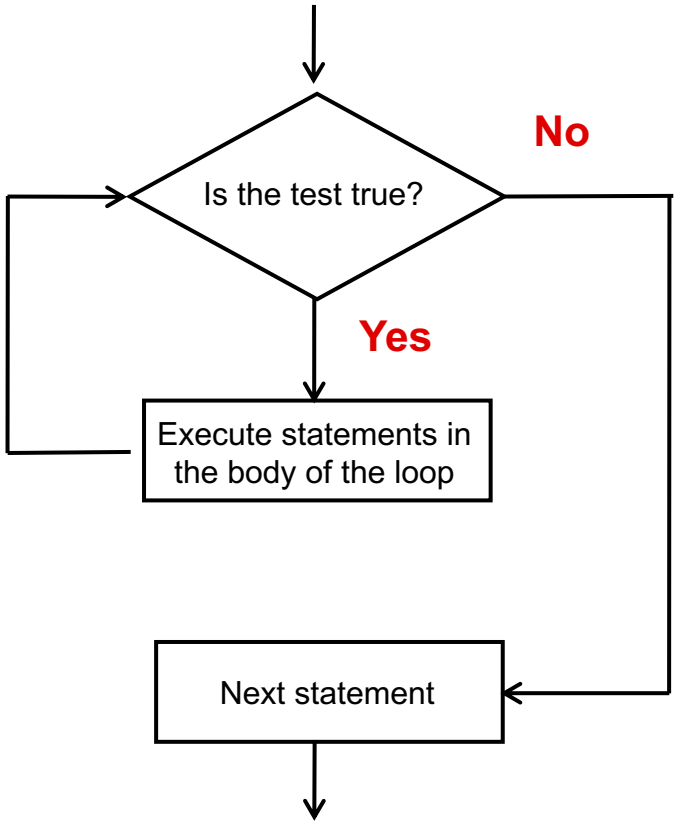
```
if (condition){
    statement(s);
}
```



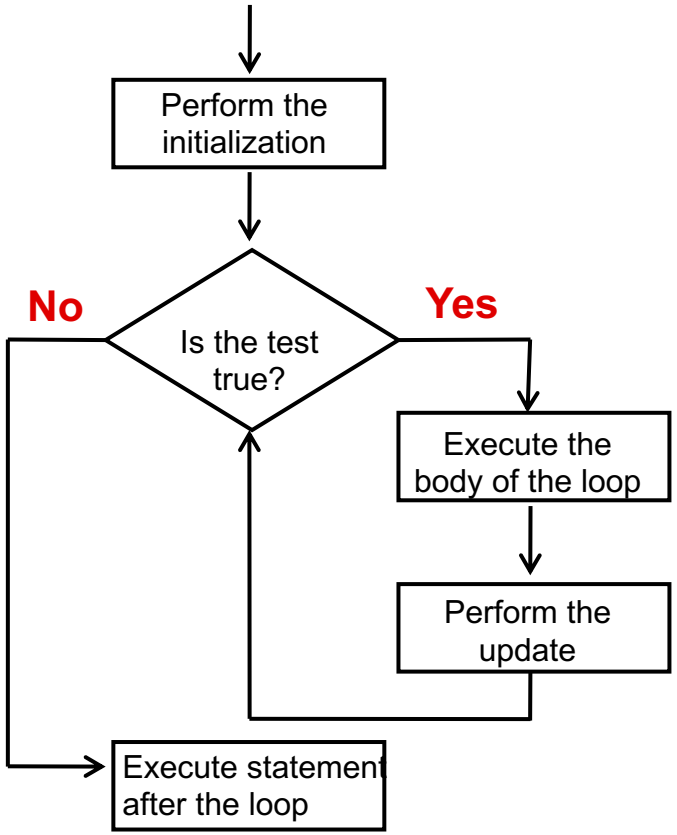

```
if (condition){
    statement(s);
}else{
    statement(s);
}
```



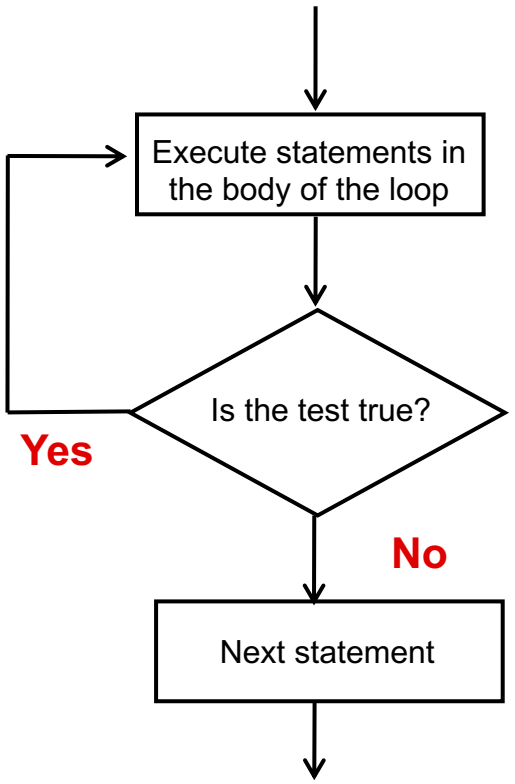
Repetition Statements




```
while (condition){
    statement(s);
}
```




```
for (initialization; condition; increment){
    statement(s);
}
```




```
do{
    statement(s);
}while(condition);
```

Methods

Methods

- A program that provides some functionality can be long and contain many statements
- A method groups a sequence of statements and should provide a well-defined, easy-to-understand functionality
- A method can take input, perform actions, and produce output

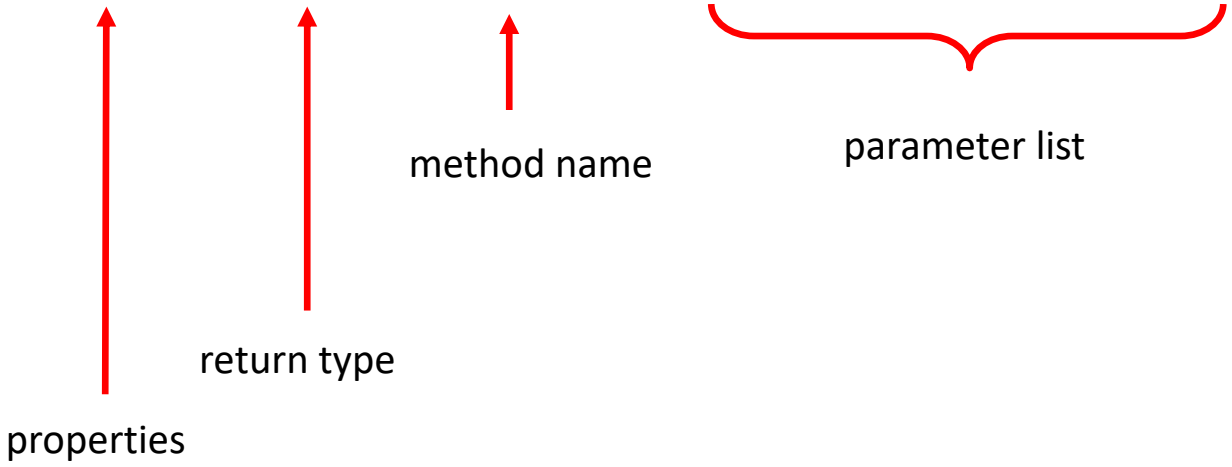
Method Declarations

- A *method declaration* specifies the code that will be executed when the method is invoked (or called)
- When a method is invoked, the flow of control jumps to the method and executes its code
- When complete, the flow returns to the place where the method was called and continues
- The invocation may or may not return a value, depending on how the method is defined

Method Header

- A method declaration begins with a *method header*

```
public static  int    add   ( int num1, int num2 )
```



- The parameter list specifies the type and name of each parameter
- The name of a parameter in the method declaration is called a *formal argument*
- **static** indicates a *static* or an *object/instance* method
- A method that is not static, is an instance method

Static Vs Instance Methods

- Static methods
  - There is one per class
- Instance methods
  - There is one per object of the class
- Static methods can not call instance methods

```
public class Car{
…
?? float km2Miles(float km)
?? float getOdometerMiles()

}
```

Java Constants

```
public static final int ARRAY_SIZE = 25;
```

```
private static final String URL = "tsekourakis.github.io";
```

- Constants in Java have to be initialized when declared!
- After that, they are read only.

Method Body

- A method header is followed by the *method body*

```
public static int add( int num1, int num2 ){
        int result = 0;
        result = num1 + num2;
        return result;
}
```

return statement

- The return expression must be consistent with the return type
- The variable result is a local variable. It is created each time the method is called, and is destroyed when it finishes executing

The `return` statement

- The `return` statement sends out a value as the result of a method

```
return expression;
```

- The *return type* of a method indicates the type of value that the method sends back to the calling location
- A method that does not return a value has a `void` return type

Parametrization

- A *parameter* is a special type of variable that allows us to pass information into a method
- A method can accept multiple parameters (separated by ,) including none
- Each time a method is called, the *actual parameters* in the invocation are copied into the formal

Declaration syntax

```
public static  int    add   ( int num1, int num2 )
```

formal parameters

Call syntax

```
add (5, 9);
```

actual parameters