

# Introduction to Programming

*with Java, for Beginners*

## Static Methods

## Motivation for Methods

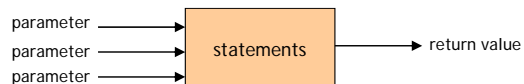
- Break up a complex problem into simpler sub-problems, which you can solve separately
  - E.g. Chocolate cake dessert
    - Baking a cake & preparing the Icing
- Write once and reuse
  - This is an application of the **DRY** principle ("Don't Repeat Yourself")
- Methods are also known as procedures, subroutines, functions

ESE112

1

## About methods

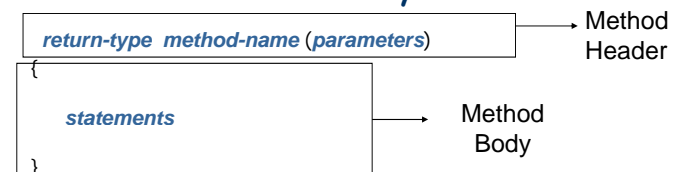
- A **method** is a named group of statements
- You execute those statements by **calling** the method
- When you call the method, you can give it **parameters** (information)
- A method typically has a **return value** (a single piece of information coming out of the method)



ESE112

2

## Method Syntax



- Example 1:

```
boolean isAdult(int age) {
    int magicAge = 21;
    return (age >= magicAge);
}
```
- Example 2:

```
double average(int a, int b) {
    int c = (a + b) / 2.0;
    return c;
}
```

ESE112

3

## Method Names

- The parts of your computation can be given a name
  - So you can reuse it later my just using the name
- Proper choice of method names is important
  - Verbs are usually best, since methods “do something”
  - Naming rules are the same for *naming variables*
    - Descriptive names make your program more readable

ESE112

4

## Parameters

- Variable(s) that get declared within parentheses of a method header
  - Each variable is associated with type
- Are inputs that will be used to do some computation within the method
- A method can have 0 or more parameters

ESE112

5

## Returning a result from a method

- If a method is to return a result, it must specify the *type* of the result:
  - `boolean` isAdult ( ...
- You must use a return statement to exit the method with a result of the correct type:
  - `return` (age >= magicAge);

```
boolean isAdult(int age) {  
    int magicAge = 21;  
    return (age >= magicAge);  
}
```

ESE112

6

## Returning *no* result from a method

- The keyword `void` is used to indicate that a method doesn't return a value
- There are two ways to indicate void method:
  - Execute a return statement by itself (no return values)
  - Reach the closing brace of the method
- Example

```
void printAge(String name, int age){  
    System.out.println(name + " is " + age + " years old.");  
    return;  
}
```

ESE112

7

## Keyword Static

- For now all methods must contain keyword **static** before the *return-type of a method*
- E.g. static boolean isAdult(int age){ ... }
- Later we will see non-static methods with Object-Oriented Programming

ESE112

8

## Accessibility Level / Modifier

- To control the usage Methods (and Classes)
  - Who has access?
- **public**: makes the method accessible from outside the class  
E.g. public static void main (String [] args)
- **private**: not accessible outside the class
- **In default case**: ( i.e. no mention of public and private): accessible if within same directory
- Accessibility\_level *appears before*
  - **static** keyword for static method
  - **class** keyword for class description

ESE112

9

## Methods within Classes

- Methods are always written within a class
- E.g. In Circle.java

```
public class Circle{
    public static double area (double radius) {
        final double PI = 3.14;
        return radius * radius * PI;
    }
}
```

ESE112

10

## Calling or Invoking a Static Method

- A way to use the method as part of an expression
- Within the *same class*:  
*staticMethodName(parameters)*
- Examples:
  1. double a = area(3.0);
  2. double x = 5.5;  
double a = area(x);  
System.out.println(a);

ESE112

11

## Calling or Invoking a Static Method

- Outside class in which it is declared in:  
***Classname.staticMethodName(parameters);***
- Examples
  1. double a = Circle.area(3.0);
  2. double a = Circle.area(x);

ESE112

12

## Main Method

- ```
public static void main(String [] args)
```
- A special static method
    - Whose return type is **void** and
    - Input is a **String array** ([ ]) (more or arrays later)
  - Entry point of a java program i.e. where the instructions starts to get executed step by step
    - If there is variable declared, then space is allocated in memory
    - If it comes across method call, then method declaration and statements are executed
    - Until the last statement, after which terminates the program

ESE112

13

## Option 1 w/ Main Method

```
public class Circle {  
    public static void main(String [] args) {  
        double a = area(3.0);  
        System.out.println("Area = " + a);  
    }  
    public static double area (double radius) {  
        final double PI = 3.14;  
        return radius * radius * PI;  
    }  
} //end of Circle class
```

ESE112

14

## Option 2 w/ Main Method

```
//In TestCircle.java  
public class TestCircle {  
    public static void main(String [] args) {  
        double a = Circle.area(3.0);  
        System.out.println("Area = " + a);  
    }  
} // end of TestCircle class  
  
//In Circle.java  
public class Circle {  
    public static double area (double radius) {  
        final double PI = 3.14;  
        return radius * radius * PI;  
    }  
} //end of Circle class
```

ESE112

15

## sumOdd: Implementation 1

```
public static int sumOdd(int n){
    //sum positive odd numbers upto n
    // e.g. sum(3) = 4
    int sum = 0;
    for (int i = 1; i <= n; i = i + 2){
        sum = sum + i;
    }
    return sum;
}
```

ESE112

16

## sumOdd: Implementation 2

```
public static int sumOdd(int n){
    //sum positive odd numbers upto n
    // e.g. sum(3) = 4
    int sum = 0;
    for (int i = 1; i <= n; i = i + 1){
        if(isOdd(i)) {
            sum = sum + i;
        }
    }
    return sum;
}
```

ESE112

17

## Return statements in loops

- Return statement should last statement before ending method
- Having return statements within loops will cause compiler to throw syntax error
  - This because the compiler does not know whether the statement is reachable or not
  - Always use variable to store the value and then finally return that value

ESE112

18