Computer Science 101 S1

Lecture 9

Contents

Methods
Scope of variables
Reasons for using methods

Coursebook: §10

Review

e
0

Methods and parameters – what is the output?

```
public class MyProgram {
      public void start() {
            int x = 0;
            String welcome = "Hello";
            myMethod(x, welcome);
            System.out.println(x);
      }
      private void myMethod(int x, String s) {
            x = x + 1;
            System.out.println(s.charAt(x));
      }
}
```

0 "Hello"

Local Variables

Variables declared **inside** a method

Local variables are created when the method is called, and they are destroyed when method ends

```
private void test() {
      int age = 10;

}

public void start() {
      test();
      System.out.println("age = " + age);
}
```

Variable 'age' is destroyed when the method ends

Illegal

Local Variables

Scope means "*which lines of code can use the variable*"

```
public void start() {
      System.out.println("Start method");

      int a;
      a = 7;

      System.out.println("a = " + a);
}
```

local variables may be used from the point they are declared until the end of the method

```
public void start() {
      System.out.println("Start method");

      System.out.println("a = " + a);

      int a;
      a = 7;
}
```

Illegal

Same identifier, different methods

Local variables can only be "seen" inside the method in which they are declared. Methods can use the same identifier for local variables

```
public void start() {
      int age = 0;
      testing01();
      testing02();
      System.out.println(age);
}
private void testing01() {
      int age = 10;
}
private void testing02() {
      int age = 20;
}
```

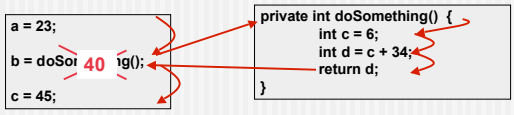
0

7

Method Execution

When a method call is made, code execution jumps to the method

If the method returns a value. When the method reaches a return statement, code execution returns to the **point of call** and the **method call** is replaced by the value returned.

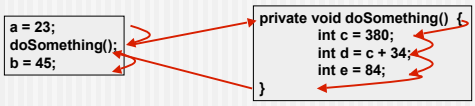


8

Method Execution

When a method call is made, code execution jumps to the method

When the method finishes executing, code execution returns to the **point of call**



Ex01 – What is the output?

4. 5
1. 5
3. 15
2. 5

```
public void start() {
      int years = 5;
      test1(years);
      System.out.println("1. " + years);
      test2(years);
      System.out.println("2. " + years);
}
private void test2(int years) {
      years = years + 10;
      System.out.println("3. " + years);
}
private void test1(int years) {
      System.out.println("4. " + years);
      years = 20;
}
```

10

Reasons for Using Methods

Ideally when we write code we would like to achieve the following:

Modularity. Divide up the project into logical parts. A large task can be broken up into a number of smaller tasks.

Avoiding Code Duplication. Rather than cutting and pasting multiple copies of code within the program, only write the code once, but call the code as many times as you need it from elsewhere in the code.

Code Reuse. Call a method that has already been written and tested to accomplish a task rather than rewrite the code again.

Easy Testing. Make the code easier to test. Once a method has been tested we can use it over and over again knowing that it works correctly.

11

Example - Guess the word

```
>java GuessApp
ANAGRAM TO GUESS
****************
     OUCAPTS
****************
Enter guess: caputos
OCTAPUS, your guess: CAPUTOS, good guess: false

>java GuessApp
ANAGRAM TO GUESS
****************
     PCTOASU
****************
Enter guess: octapus
OCTAPUS, your guess: OCTAPUS, good guess: true
```

12

Example - Guess the word

It is a good idea to divide large tasks into a number of smaller tasks. In this way each task is a single small programming task which is clear and easy to write.

Think about the tasks involved in writing the code for the programming assignment from the previous slide.

To solve the programming problem on the previous slide I have written eight java statements shown on the next slide. Put the statements in the correct order in the start() method.

13

Ex02 - Guess the word

```
1. userGuess = getUserGuess("Enter guess");
2. checkGuess = checkUserGuess(theWord, userGuess);
3. theWord = getRandomWord();
4. displayResult(theWord, userGuess, checkGuess);
5. String theWord, anagramWord, userGuess;
6. displayAnagram(anagramWord);
7. boolean checkGuess;
8. anagramWord = createAnagram(theWord);
```

```
public class GuessProgram {
 public void start() {


 }
       …
}
```

14

Code Reuse

Once a method has been tested the method can be used in any program which needs that task done.

```
public class Game {
 public void start() {
  int row = getUserInt("Enter row");
  int col = getUserInt("Enter column");
  …
 }
 private int getUserInt(String prompt) {
  System.out.print(prompt + " ");
  String userStr = Keyboard.readInput();
  int userIn = Integer.parseInt(userStr);
  return userIn;
 }
}
```

```
public class NumberGuessing {
 public void start() {
  int userN = getUserInt("Enter guess");
  int compN = (int)(Math.random() * 10);
  …
 }
 private int getUserInt(String prompt) {
  System.out.print(prompt + " ");
  String userStr = Keyboard.readInput();
  int userIn = Integer.parseInt(userStr);
  return userIn;
 }
}
```

15

Avoid Code Duplication

```
public class Game {
 public void start() {
  int row, col, number, steps;
  String userStr;
  System.out.print("Enter row: ");
  userStr = Keyboard.readInput();
  row = Integer.parseInt(userStr);

  System.out.print("Enter col: ");
  userStr = Keyboard.readInput();
  col = Integer.parseInt(userStr);

  System.out.print("Enter number: ");
  userStr = Keyboard.readInput();
  number = Integer.parseInt(userStr);

  System.out.print("Enter steps: ");
  userStr = Keyboard.readInput();
  steps = Integer.parseInt(userStr);
  …
 }
}
```

```
public class Game {
 public void start() {
  int row = getUserInt("Enter row");
  int col = getUserInt("Enter column");
  int num = getUserInt("Enter number");
  int steps = getUserInt("Enter steps");
  …
 }
 private int getUserInt(String prompt){
  System.out.print(prompt + " ");
  String userStr = Keyboard.readInput();
  int userIn = Integer.parseInt(userStr);
  return userIn;
 }
}
```

16

Methods are easy to test

```
public class Game {
 public void start() {
  String word = "OCTAPUS";
  displayAnagram(word);
  word = "A VERY LONG STRING WHICH I WANT TO DISPLAY";
  displayAnagram(word);
  word = "";
  displayAnagram(word);
 }
 private void displayAnagram(String hiddenWord) {
   String stars = "****************************************";
   int len = hiddenWord.length();
   System.out.println("ANAGRAM TO GUESS");
   System.out.println(stars.substring(0, len+10));
   System.out.println("      " + hiddenWord);
   System.out.println(stars.substring(0, len+10));
}}
```

17

Parameter names

Do both the following programs behave in the same way?

```
public class Game {
 public void start() {
  int row = getUserInt("Enter row ");
  int col = getUserInt("Enter column ");
  int num = getUserInt("Enter number ");
  int steps = getUserInt("Enter steps ");
  …
 }
 private int getUserInt(String words){
  System.out.print(words);
  String userStr = Keyboard.readInput();
  int userIn = Integer.parseInt(userStr);
  return userIn;
 }
}
```

```
public class Game {
 public void start() {
  int row = getUserInt("Enter row ");
  int col = getUserInt("Enter column ");
  int num = getUserInt("Enter number ");
  int steps = getUserInt("Enter steps ");
  …
 }
 private int getUserInt(String prompt){
  System.out.print(prompt);
  String userStr = Keyboard.readInput();
  int userIn = Integer.parseInt(userStr);
  return userIn;
 }
}
```

18

Order of methods

Do both the following programs behave in exactly the same way?

```
public class Game {
 private int getUserInt(String words){
  System.out.print(words);
  String userStr = Keyboard.readInput();
  int userIn = Integer.parseInt(userStr);
  return userIn;
 }
 public void start() {
  int row = getUserInt("Enter row ");
  int col = getUserInt("Enter column ");
  int num = getUserInt("Enter number ");
  int steps = getUserInt("Enter steps ");
  …
 }
}
```

```
public class Game {
 public void start() {
  int row = getUserInt("Enter row ");
  int col = getUserInt("Enter column ");
  int num = getUserInt("Enter number ");
  int steps = getUserInt("Enter steps ");
  …
 }
 private int getUserInt(String words){
  System.out.print(words);
  String userStr = Keyboard.readInput();
  int userIn = Integer.parseInt(userStr);
  return userIn;
 }
}
```

19

Ex03 - Why do I get an error?

```
public class ProblemCode3 {
 public void start() {
    double x = 5;
    int y = 7;
    x = method1(x, y);
 }
 private int method1(int xVal, int yVal) {
    xVal = xVal * 2;
    return yVal + xVal;
 }
}
```

20

Ex04 - Why do I get an error?

```
public class ProblemCode4 {
 public void start(){
    final int fixed_Price = 35;
    System.out.println("Fixed price: $" + fixed_Price);
    method1();
 }
 private void method1(){
    System.out.println("Fixed price: $" + fixed_Price);
 }
}
```

21

Ex05 - Why do I get an error?

```
public class ProblemCode5 {
 public void start(){
    int price = 35;
    System.out.println("Fixed price: $" + price);
    price = 45;
    int result = method1(price );
 }
 private void method1(int fPrice){
    System.out.println("Fixed price: $" + fPrice);
 }
}
```

22

Ex06 - Why do I get an error?

```
public class ProblemCode6 {
 public void start(){
  final int FIXED_PRICE = 35;
    boolean result = method1(FIXED_PRICE);
 }
 private boolean method1(int num){
    return num = 35;
 }
}
```

23

What you need to know

- The scope of a variable.
- Code execution.
- Modularity, code reuse, avoid code duplication, methods are easy to test.
- Methods in a class can be defined in any order.