CS 101 Computer Programming

Methods (cont’d..)

Özyeğin University

slides partially adapted from UW CSE 142 course slides
New Concepts

• Two new concepts:
  – **Return** values
  – **Parameters**

• The simple method in our example did not have any return value or a parameter

```java
public void printSeparator() {
    println("***************");
    println("***************");
}
```
New Concepts

• Two new concepts:
  – **Return** values
  – **Parameters**

• The simple method in our example did not have any return value or a parameter

```java
public void printSeparator() {
    println("***************");
    println("***************");
}
```

First we will focus on this new concept
Refresher: Printing a Separator

• The original program within the run method called this method to print separator lines

/* print a separator line on output */
public void printSeparator() {
    println("***************");
    println("***************");
}
Refresher: Example

```java
public void run() {
    /* produce some output */
    ...
    printSeparator();
    /* produce some other output */
    ...
    printSeparator();
    /* produce even more output */
    ...
    printSeparator();
    /* produce the final output */
    ...
}
```

The code named `printSeparator`

```java
println("************************");
println("************************");
```
Refresher: Example

```java
public void run() {
    /* produce some output */
    ...
    printSeparator();
    /* produce some other output */
    ...
    printSeparator();
    /* produce even more output */
    ...
    printSeparator();
    /* produce the final output */
    ...
}
```

The code named `printSeparator`

```
println("***************");
println("***************");
```
Refresher: Example

```java
public void run() {
    /* produce some output */
    ...
    printSeparator();
    /* produce some other output */
    ...
    printSeparator();
    /* produce even more output */
    ...
    printSeparator();
    /* produce the final output */
    ...
}
```

The code named `printSeparator`

```java
println("***************");
println("***************");
```
Refresher: Example

```java
public void run() {
    /* produce some output */
    ...
    printSeparator();
    /* produce some other output */
    ...
    printSeparator();
    /* produce even more output */
    ...
    printSeparator();
    /* produce the final output */
    ...
}
```

The code named `printSeparator`

```java
println("***************");
println("***************");
println("***************");
println("***************");
println("***************");
```
A new problem..

- The client wants another change
  - The program should print out 5 rows of stars when it starts and when it ends
  - But it should print out 2 rows of stars in-between messages
One possible solution

- Define two different methods for two different type of separators

```java
/* print 2 lines of separator line on output */
public void printSeparator2Lines() {
    println("***************");
    println("***************");
}

/* print 5 lines of separator line on output */
public void printSeparator5Lines() {
    for(int i = 0; i < 5; i++) {
        println("***************");
    }
}
```
public void run() {
    printSeparator5Lines();
    /* produce some output */
    ...
    printSeparator2Lines();
    /* produce some other output */
    ...
    printSeparator2Lines();
    /* produce even more output */
    ...
    printSeparator2Lines();
    /* produce the final output */
    ...
    printSeparator5Lines();
}
Can we reuse the same method for two different type of separators

- How can we generalize the required function
  - Print two rows of stars
  - Print N rows of stars

- N is the number of rows we want to print
- N is the information that method needs to know
The modified Example

```java
public void run() {
    printSeparator(5);
    /* produce some output */
    ...
    printSeparator(2);
    /* produce some other output */
    ...
    printSeparator(2);
    /* produce even more output */
    ...
    printSeparator(2);
    /* produce the final output */
    ...
    printSeparator(5);
}
```

code for printSeparator
public void run() {
    printSeparator(5);
    /* produce some output */
    ...
    printSeparator(2);
    /* produce some other output */
    ...
    printSeparator(2);
    /* produce even more output */
    ...
    printSeparator(2);
    /* produce the final output */
    ...
    printSeparator(5);
}
Structure of the modified method

/* print a separator line on output */
public void printSeparator(int n) {
    /* code for the method */
}

• n is called the \textbf{argument} (or parameter) of the method
• n can be used inside the method like a variable
Code for the modified method

/* print a separator line on output */
public void printSeparator(int n) {
    for(int i = 0; i < n; i++) {
        println("***************");
    }
}
}
Need for multiple arguments

• What if we want to set both the **number of lines** and the **number of stars per line**?

```java
/* print a separator line on output */
public void printSeparator(int n) {
    for(int i = 0; i < n; i++) {
        println("***************");
    }
}
```
Multiple arguments

• A method can have more than one argument
• Arguments are matched based on **order** and **type**

```java
printSeparator(10, 3);
/* print a separator line on output */
public void printSeparator(int s, int n) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < s; j++) {
            print("*");
        }
    }
    println("\n");
}
```
Multiple arguments

• A method can have more than one argument
• Arguments are matched based on order and type

printSeparator(10, 3);

/* print a separator line on output */
public void printSeparator(int s, int n) {
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < s; j++) {
            print("*");
        }
    }
    println(" ");
}

Exercise: methods and parameters

• Draw the following text on the screen
Did you notice any reusable elements?
Potential Reusable Elements

• To draw A, we can first draw P, and draw only one additional line
• To draw R, we can first draw P, and draw only one additional line
• To draw E, we can first draw L, and draw only two additional lines
The power of methods for reuse

• We will use methods to exploit common elements in the drawing and write code only once for these elements
public void run() {
    drawP(20, 50);
    drawA(70, 50);
    drawR(120, 50);
    drawA(170, 50);
    drawL(220, 50);
    drawL(270, 50);
    drawE(320, 50);
    drawL(370, 50);
    
    public void drawP(int x, int y) {
        ....
    }
    public void drawA(int x, int y) {
        drawP(x, y);
        ....
    }
    public void drawR(int x, int y) {
        drawP(x, y);
        ....
    }
    public void drawL(int x, int y) {
        ....
    }
    public void drawE(int x, int y) {
        drawL(x, y);
        ...
    }
}
Solution for the Exercise

• Solution and the explanations are available on the Web: https://vimeo.com/32473689 (key: cs101)
Method call mechanism

• Each method can be considered as a frame that contains
  – the code of the method
  – memory cells for all the parameters
  – memory cells for all the variables declared inside the method
Method call mechanism

- Calling a method brings its frame to the top of the execution stack.
- Execution initially starts with the frame of the `run()` method on the top.
- The top frame is the active method that executes. The frames below the top one are passive; they wait to become the top frame again.
At the end of execution..

- When the top method terminates (i.e. *returns*), its frame is kicked out. The frame below becomes the top frame again, and resumes execution.
Example: Drawing rectangles

- Task: Drawing 3 rectangles on the screen, one red, one blue, and one green
public class RedRectangle extends GraphicsProgram {
    public void run() {
        drawRedRectangle();
        drawBlueRectangle();
    }

    public void drawRedRectangle() {
        GRect rect = new GRect(100, 20, 60, 80);
        rect.setColor(Color.RED);
        rect.setFilled(true);
        add(rect);
    }

    public void drawBlueRectangle() {
        GRect rect = new GRect(20, 120, 80, 40);
        rect.setColor(Color.BLUE);
        rect.setFilled(true);
        add(rect);
        drawGreenRectangle();
    }

    public void drawGreenRectangle() {
        GRect rect = new GRect(150, 50, 60, 20);
        rect.setColor(Color.GREEN);
        rect.setFilled(true);
        add(rect);
    }
}

public void run() {
    drawRedRectangle();
    drawBlueRectangle();
}

public void drawRedRectangle() {
    GRect rect = new GRect(100, 20, 60, 80);
    rect.setColor(Color.RED);
    rect.setFilled(true);
    add(rect);
}

public void drawBlueRectangle() {
    GRect rect = new GRect(20, 120, 80, 40);
    rect.setColor(Color.BLUE);
    rect.setFilled(true);
    add(rect);
    drawGreenRectangle();
}

public void drawGreenRectangle() {
    GRect rect = new GRect(150, 50, 60, 20);
    rect.setColor(Color.GREEN);
    rect.setFilled(true);
    add(rect);
}
```java
public void run() {
    drawRedRectangle();
    drawBlueRectangle();
}
```
public void run() {
    drawRedRectangle();
    drawBlueRectangle();
}

public void drawRedRectangle() {
    GRect rect = new GRect(100, 20, 60, 80);
    rect.setColor(Color.RED);
    rect.setFilled(true);
    add(rect);
}
```java
public void run() {
    drawRedRectangle();
    drawBlueRectangle();
}

public void drawRedRectangle() {
    GRect rect = new GRect(100, 20, 60, 80);
    rect.setColor(Color.RED);
    rect.setFilled(true);
    add(rect);
}
```
public void run() {
    drawRedRectangle();
    drawBlueRectangle();
}
public void run() {
    drawRedRectangle();
    drawBlueRectangle();
}

public void drawBlueRectangle() {
    GRect rect = new GRect(20, 120, 80, 40);
    rect.setColor(Color.BLUE);
    rect.setFilled(true);
    add(rect);
    drawGreenRectangle();
}
public void run() {
    public void drawBlueRectangle() {
        GRect rect = new GRect(20, 120, 80, 40);
        rect.setColor(Color.BLUE);
        rect.setFilled(true);
        add(rect);
        drawGreenRectangle();
    }
}
public void run() {
    public void drawBlueRectangle() {
        GRect rect = new GRect(20, 120, 80, 40);
        rect.setColor(Color.BLUE);
        rect.setFilled(true);
        add(rect);
        drawGreenRectangle();
    }
}
public void run() {
    drawRedRectangle();
    drawBlueRectangle();
}

public void drawBlueRectangle() {
    GRect rect = new GRect(20, 120, 80, 40);
    rect.setColor(Color.BLUE);
    rect.setFilled(true);
    add(rect);
    drawGreenRectangle();
}

public void drawGreenRectangle() {
    GRect rect = new GRect(150, 50, 60, 20);
    rect.setColor(Color.GREEN);
    rect.setFilled(true);
    add(rect);
}
public void run() {
    public void drawBlueRectangle() {
        public void drawGreenRectangle() {
            GRect rect = new GRect(150, 50, 60, 20);
            rect.setColor(Color.GREEN);
            rect.setFilled(true);
            add(rect);
        }
    }
}

```java
public void run() {
    public void drawBlueRectangle() {
        GRect rect = new GRect(20, 120, 80, 40);
        rect.setColor(Color.BLUE);
        rect.setFilled(true);
        add(rect);
        drawGreenRectangle();
    }
}
```
public void run() {
    drawRedRectangle();
    drawBlueRectangle();
}
Example: Drawing rectangles

• Using just one method with arguments

```java
public void run() {
    drawRectangle(100, 20, 60, 80, Color.red);
    drawRectangle(20, 120, 80, 40, Color.blue);
    drawRectangle(150, 50, 60, 20, Color.green);
}

public void drawRectangle(int x, int y, int w, int h, Color c) {
    GRect rect = new GRect(x, y, w, h);
    rect.setColor(c);
    rect.setFilled(true);
    add(rect);
}
```

pay attention to the order and type of arguments
Previously on CS101

• Method: a named block of statements which we can invoke.

```java
public void drawRedRectangle() {
    GRect rect = new GRect(100, 20, 60, 80);
    rect.setColor(Color.RED);
    rect.setFilled(true);
    add(rect);
}
```
Previously on CS101

- Method: can be parameterized.
- A parameter is like a variable for which the caller has to provide a value.

```java
    public void run() {
        drawRedRectangle(100, 20);
        drawRedRectangle(180, 90);
        drawRedRectangle(400, 140);
    }

    public void drawRedRectangle(int x, int y) {
        GRect rect = new GRect(x, y, 60, 80);
        rect.setColor(Color.RED);
        rect.setFilled(true);
        add(rect);
    }
```
CD Player is a parameterized method

Thanks to Mehran Sahami for this analogy.
CD Player is a parameterized method
CD Player is a parameterized method
Invoking/calling the method

Must press the play button
Facebook is a parameterized method
Garth Vader: Sorry bout your hand ... it's easily fixed
6 minutes ago · Comment: Like · See Walk-to-Wall
Luke Skywalker: I blew up your hood, so we're even ... UGH
4 minutes ago · Delete
Write a comment...

Garth Vader and Princess Leia are now friends.
6 minutes ago · Comment: Like

Garth Vader and Princess Leia are now friends with Dan Solo.
3 minutes ago

Princess Leia: Meeting up with my Boo later, can't wait. Love you Hanny!
3 minutes ago · Comment: Like

Luke Skywalker: Intense workout, that Bikram Yoda kicked my butt today
24 minutes ago · Comment: Like

Joda Masters: Stop being such a daddy's boy
13 minutes ago · Delete

Luke Skywalker: What did you say about my Daddy?
13 minutes ago · Delete

Garth Vader: You know it to be true
13 minutes ago · Delete
Write a comment...

Joda Masters: Suck it up pretty boy
19 minutes ago · Comment: Like · Share

Luke Skywalker: At Bikram Yoda Class
20 minutes ago · Comment: Like
Facebook is a parameterized method
Superman’s social network nightmare.
Invoking/calling the method

Must press the login button
println is a parameterized method

println("Hello, world!");
println is a parameterized method

println("May the force be with you!");
Invoking the method

```java
public void run() {
    println("May the force be with you!");
}
```
New Concepts

- Two new concepts:
  - **Return** values
  - **Parameters**

- The simple method in our example did not have any return value or a parameter

```java
public void printSeparator() {
    println("***************");
    println("***************");
}
```

Now we will focus on this new concept
A new example problem..

• Write a method which, given the radius, computes and returns the area of a circle with that radius

• The new problem here is that
  – The method should return a value
Return values..

• Arguments are used for sending data to the method

• Return values are used for the opposite: for sending data back
Example: The *area* method

- Write a method which, given the radius, computes and returns the area of a circle with that radius
- New features:
  - The return statement sends a value back
  - The type of the returned value is stated before the method name

```java
/* calculates the area of a circle with radius r */
public double areaOfCircle(double r) {
    double area = 3.14 * r * r;
    return area;
}
```
Control and Data Flow

```java
public void run() {
    double r = readDouble("Enter the radius of the circle");
    double a = areaOfCircle(r);
    println("The area of the circle is "+ a);
}

public double areaOfCircle(double radius) {
    double area = 3.14 * radius * radius;
    return area;
}
```
Returning a value to the caller

• The `return` statement...

Which one’s heavier?
Returning a value to the caller

• The **return** statement...

Calls Asterix.
passes him the arguments
Returning a value to the caller

- The return statement...

Performs measurement

Calls Asterix.

waiting for Asterix
Returning a value to the caller

• The **return** statement...

Calls Asterix.

returns the result
Returning a value to the caller

- The **return** statement...

continues his life...
Different parameters

• May do the same thing by passing Asterix different arguments.
Returning a value to the caller

- The `return` statement...

Calls Asterix.

`returns` the result
Returning a value to the caller

- A method can return a value to its caller.

```java
public void run() {
    int n1 = readInt("Enter first number: ");
    int n2 = readInt("Enter second number: ");
    int n3 = readInt("Enter third number: ");

    int maxNumber = max(max(n1,n2), n3);
    println("Max is "+ maxNumber);
}

public int max(int x, int y) {
    if(x > y) {
        return x;
    } else {
        return y;
    }
}
```
Exercise: Methods

• Write a method that calculates the factorial of a number, n
• n should be provided as an integer argument
• The method will be used by the run method as follows

```java
import acm.program.*;

public class Methods extends ConsoleProgram {

    public void run() {
        int number = readInt("Enter a number: ");
        int result = factorial(number);
        println(number + "! = " + result);
    }

    /* write the method here */
}
```
Exercise: Methods

import acm.program.*;
public class Methods extends ConsoleProgram {
    public void run() {
        int number = readInt("Enter a number: ");
        int result = factorial(number);
        println(number + "! = " + result);
    }

    /* calculates the factorial of a number*/
    public int factorial(int n) {
        int fact = 1;
        for (int i = 2; i <= n; i++) {
            fact *= i;
        }
        return fact;
    }
}

/* calculates the factorial of a number*/
public int factorial(int n) {
    int fact = 1;
    for (int i = 2; i <= n; i++) {
        fact *= i;
    }
    return fact;
}
Local variables

• Variables that are defined inside a method
• Cannot be used by other methods
  – The value of a local variable can be sent to another method only as an argument of that method
• Created just before the method executes, destroyed when the method returns
  – Local to the method where it is defined
  – Note: parameters are also local
Method Scopes

• Methods have their own scopes. The names of the parameters may or may not be the same as the local variables in the caller.
Transfering arguments..

• When a method’s frame is put on top of the stack, the cells of its parameters are filled with the copies of the values of the arguments from the caller site.

• Likewise, a method’s return value is copied to its caller, who has been waiting.
More on return..

• For **void** methods
  – `return;`
  – Causes the method to stop execution and return back to the point of call
  – **Optional**: method automatically returns when all the statements in the method code block are executed

• For methods that return a value
  – `return expression;`
  – Causes the method to stop execution and return back to the point of call
  – The method call is replaced with the returned expression
  – **Required**
public class PrimeChecker extends ConsoleProgram {
    public void run() {
        int N = readInt("Enter a positive integer: ");
        checkPrimity(N);
    }

    public void checkPrimity(int number) {
        if (number == 1) {
            println(number + " is not a prime number");
            return; // Terminate immediately
        }

        for (int i = 2; i < number; i++) {
            if (number % i == 0) {
                println(number + " is not a prime number");
                return; // Terminate immediately
            }
        }

        println(number + " is a prime number");
    }
}
Review Method Control Flow

• Methods allow you to visit a block of code and then come back
  • This code block can be elsewhere in your program (in another class)

• We have described the basic flow before..

![Diagram of method call and return]
When a method is called..

- Memory space is allocated for the method arguments and local variables
- Argument values are copied
- Control transfers to the method
- The method executes
- Control and the return value is transferred back to the point of call
Exercise: raise to power

• Write a method that calculates $n^k$

```java
import acm.program.*;

public class Methods extends ConsoleProgram {

    public void run() {
        int pow = raiseToPower(2, 3);
        println("2 to the power 3 is " + pow);
    }

    /* write the method here */
}
```
Exercise: raise to power

import acm.program.*;
public class Methods extends ConsoleProgram {
    public void run() {
        for(int i = 0; i <= 10; i++) {
            int pow = raiseToPower(2, i);
            println("2 to the power " + i + " is " + pow);
        }
    }
    public int raiseToPower(int n, int k) {
        // Compute and return n to the power k
        int power = 1;
        for(int i=0; i < k; i++) {
            power *= n;
        }
        return power;
    }
}

CS 101 | Özyeğin University
Alternatively

```java
public class PowersOfTwo extends ConsoleProgram {
    public void run() {
        for(int i = 0; i <= 10; i++) {
            int n = readInt("Enter base:");
            int k = readInt("Enter exponent:");
            int pow = raiseToPower(n, k);
            println(n + " to the power " + k + " is " + pow);
        }
    }

    public int raiseToPower(int n, int k) {
        // Compute and return n to the power k
        int power = 1;
        for(int i=0; i < k; i++) {
            power *= n;
        }
        return power;
    }
}
```
Exercise: digital root

- Write a method that calculates the digital root of a number

```java
import acm.program.*;

public class Methods extends ConsoleProgram {

    public void run() {
        int n = readInt("Enter a positive integer: ");
        int root = digitalRoot(n);
        println("Your number's digital root is " + root);
    }

    /* write the method(s) here */
}
```
import acm.program.*;

public class Methods extends ConsoleProgram {

    public void run() {
        int n = readInt("Enter a positive integer: ");
        int root = digitalRoot(n);
        println("Your number's digital root is "+root);
    }

    public int digitalRoot(int number) {
        int dsum = sumDigits(number);
        while (dsum >= 10) {
            dsum = sumDigits(dsum);
        }
        return dsum;
    }

    public int sumDigits(int n) {
        int sum = 0;
        while (n > 0) {
            int lastDigit = n % 10;
            sum += lastDigit;
            n = n / 10;
        }
        return sum;
    }
}

Methods: Summary

• Methods may take several parameters, or none
• Methods may return one value, or none
• Methods are valuable
  – A tool for program structuring
  – Provide abstract services: the caller cares what the methods do, but not how
  – Makes programs easier to write and understand
Coding Style: comments for methods

• The comment above a method must give a complete specification of what the method does, including the significance of all arguments and any returned value.
• Someone wishing to use the method should be able to cover the method body and find everything they need to know in the method heading and comment.

    /* calculates the factorial of a number, n */
    public int factorial(int n) {
        ...
    }