Remember?

- Control Flow: the order in which statements are executed

Diagram:

[Diagram showing control flow with nodes and arrows]
Another form of 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)

• Methods are also called as functions, procedures, or subroutines
Why use methods?

• Here is an example case:
  – We want to write a program that prints out different messages on the screen,
  – and we would like to display the following pattern in-between messages to separate them
    **********************
    **********************
    **********************
Solution approach

• Use two println

println("***************");
println("***************");
The structure of the code..

public void run() {
    /* produce some output */
    ...
    println("****************");
    println("****************");
    /* produce some other output */
    ...
    println("****************");
    println("****************");
    /* produce even more output */
    ...
    println("****************");
    println("****************");
    /* produce the final output */
    ...
}
The problem

• The solution is fine
  – It does produce the desired result
  – No issues with respect to correctness and functionality

• But, there is an issue from a different perspective
  – How hard it would be to change the program in the future
  – How much work is it to write the same statements over and over
  – ...

What if..

• Later on the client wants us to change
  – The number of rows of stars
  – The number of stars per row
  – Use another character than a star
  – Print the date and time with each separator
  – ...

• How much work is involved?
If we want to change anything

• Have to edit every copy of the code in the program
• It is easy to overlook some copies
• It can be hard to find them all
  – They might not be written identically
• A piece of code that looks like serving the same purpose might be doing something else
The Big Idea behind methods

- Identify a sub-problem that has to be solved
- Write code for solving that sub-problem, only once
- Give that code a name: that makes it a method
- Whenever the sub-problem needs to be solved, use the method name to say:
  - Go to that code now to take care of this sub-problem, and do not come back until you are done!
Example:

• Identify a sub-problem that has to be solved
• Take the repeated lines of code
  ```java
  println("**************");
  println("**************");
  ```
• Wrap it as a method by giving it a name, e.g., `printSeparator`
public void run() {
    /* produce some output */
    ...
    printSeparator();
    /* produce some other output */
    ...
    printSeparator();
    /* produce even more output */
    ...
    printSeparator();
    /* produce the final output */
    ...
}

/* produce some output */
 ...
printSeparator();
/* produce some other output */
 ...
printSeparator();
/* produce even more output */
 ...
printSeparator();
/* produce the final output */
 ...

println("***************");
println("***************");

println("***************");
println("***************");

The code named printSeparator
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("***************");
```
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("***************");
Example:

```java
class Example {
    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("************************");
```
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("************************");

**Question:**

If we need to change the separator in this new version of the program, *What do we have to do?*

*How many places in the program have to be changed?*
The Big Picture so far..

• Methods
• Method control flow
• The motivation for methods

• Next..
  – How to define and use methods in Java
  – Details of different type of methods and their usage
How to define methods

• The following is a typical method declaration

```java
/* print a separator line on output */
public void printSeparator() {
    println("***************");
    println("***************");
}
```
How to define methods

• The following is a typical method declaration

```java
/* print a separator line on output */
public void printSeparator() {
    println("***************");
    println("***************");
}
```

heading comment

method name

a method can have any number of and any kind of statements
import acm.program.*;

public class Report extends ConsoleProgram {

    public void run() {
        printSeparator();
    }

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

Using a method..

import acm.program.*;

public class Report extends ConsoleProgram {

    public void run() {
        printSeparator();
    }

    public void printSeparator() {
        println("***************");
        println("***************");
    }
}
New concepts

• Two new concepts that will be discussed later on..
  – Return values
  – Parameters
• The simple method in our example do not have any return value or a parameter

```java
public void printSeparator() {
    println("***************");
    println("***************");
}
```
We have already seen and used several methods:

```java
public void run() {
    ...
}

println("hello world");
```
Prewritten methods

• Prewritten methods are commonly collected in packages

• Remember `import acm.program.*;` ??
  – Tells Java that you want to use methods in this package
  – There are many other useful packages
Exercise: Methods

• Modify the program below
• Create and use a method that executes the highlighted section(s) of the code (name the method as `endMessage`)
• Observe the output

```java
import acm.program.*;
public class Stars extends ConsoleProgram {
    public void run() {
        println("hello world");
        println("-- end of message -- ");
        println("hello methods");
        println("-- end of message -- ");
        println("see you next time");
        println("-- end of message -- ");
    }
}
```