ÖZYEĞİN UNIVERSITY

CS105
Introduction to Object-Oriented Programming

Prof. Dr. Nizamettin AYDIN

naydin@itu.edu.tr

nizamettin.aydin@ozyegin.edu.tr

Method

Outline

- Algorithms
- Removing redundancy
- Methods
- Using methods
- Design of an algorithm
- Declaring a method
- Calling a method
- Program with a method
- Methods calling methods
- Control flow
- When to use method

ALGORITHMS

- **Algorithm**:
  - –A list of steps for solving a problem.
- Example algorithm: "Bake sugar cookies"
  - – Mix the dry ingredients.
  - – Cream the butter and sugar.
  - – Beat in the eggs.
  - – Stir in the dry ingredients.
  - – Set the oven temperature.
  - – Set the timer.
  - – Place the cookies into the oven.
  - – Allow the cookies to bake.
  - – Spread frosting and sprinkles onto the cookies.
  - – ...

PROBLEMS WITH ALGORITHMS

- **lack of structure**:
  - Many tiny steps; tough to remember.
- **redundancy**:
  - Consider making a double batch...

- Mix the dry ingredients.
- Cream the butter and sugar.
- Beat in the eggs.
- Stir in the dry ingredients.
- Set the oven temperature.
- Set the timer.
- Place the first batch of cookies into the oven.
- Allow the cookies to bake.
- Set the timer.
- Place the second batch of cookies into the oven.
- Allow the cookies to bake.
- Mix ingredients for frosting.
- ...

STRUCTURED ALGORITHMS

- **structured algorithm**: Split into coherent tasks.
  1. Make the cookie batter.
     - Mix the dry ingredients.
     - Cream the butter and sugar.
     - Beat in the eggs.
     - Stir in the dry ingredients.
  2. Bake the cookies.
     - Set the oven temperature.
     - Set the timer.
     - Place the cookies into the oven.
     - Allow the cookies to bake.
  3. Decorate the cookies.
     - Mix the ingredients for the frosting.
     - Spread frosting and sprinkles onto the cookies.

REMOVING REDUNDANCY

- A well-structured algorithm can describe repeated tasks with less redundancy.
  1. Make the cookie batter.
     - Mix the dry ingredients.
     - ...
  2. A. Bake the cookies (first batch).
     - Set the oven temperature.
     - Set the timer.
     - ...

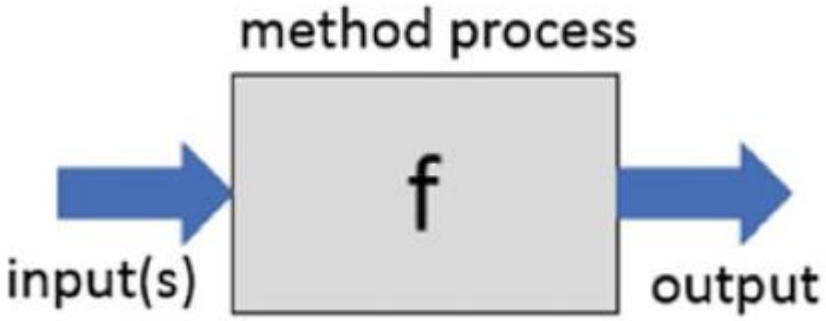
     B. Bake the cookies (second batch).
  3. Decorate the cookies.
     - ...

A PROGRAM WITH REDUNDANCY

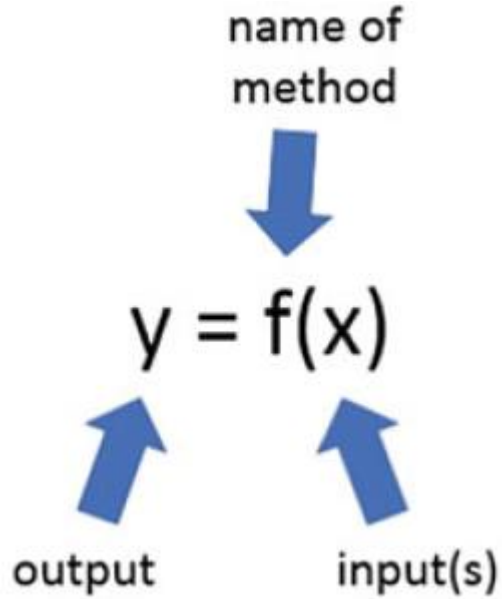
```
public class BakeCookies {
    public static void main(String[] args) {
        System.out.println("Mix the dry ingredients.");
        System.out.println("Cream the butter and sugar.");
        System.out.println("Beat in the eggs.");
        System.out.println("Stir in the dry ingredients.");
        System.out.println("Set the oven temperature.");
        System.out.println("Set the timer.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");
        System.out.println("Set the oven temperature.");
        System.out.println("Set the timer.");
        System.out.println("Place a batch of cookies into the oven.");
        System.out.println("Allow the cookies to bake.");
        System.out.println("Mix ingredients for frosting.");
        System.out.println("Spread frosting and sprinkles.");
    }
}
```

METHODS

- Methods have their origins in the mathematical concept of a function.
- A **method** is a function that belongs to a class.
  - A method performs some useful behaviour made up using code.
  - A method can have any number of inputs, and either one or zero output.
  - A method with zero outputs is called a **void method**.



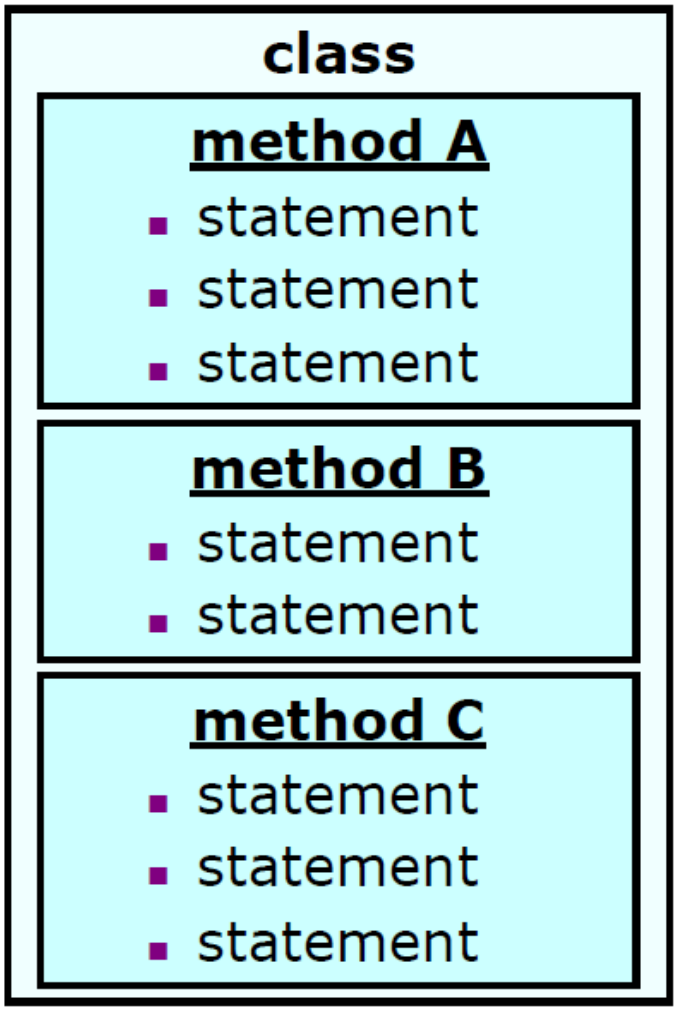
A black box model with input(s) and an output



A maths function of the form y = f(x)

METHODS

- method:
  - –A named group of statements.
    - denotes the structure of a program
    - eliminates redundancy by code reuse
- **procedural decomposition**:
  - – dividing a problem into methods
- Writing a method is like
  - – adding a new command to Java.



METHODS

- The maths way is helpful
  - –helps us understand the origins of the syntax for calling methods.
- The black box model is helpful
  - –reminds us of the procedural programming paradigm,
  - –process inside the method is encapsulated within that method.
- All methods in Java must belong to a class
  - –they cannot exist in isolation.
  - –can be defined anywhere inside their host class.
- A method definition takes the general form:

```
<access-modifier> <return-type> method-name(<formal-parameters>)
{
    // Body of method
    <return-statement-if-not-void>
}
```

METHODS

- **<Access-modifier>:**
  - public or private depending on whether the method is intended to be invokable by other classes, or only by methods inside the class in which it was defined.
  - Other access modifiers also exist.
  - If not specified, the access modifier is assumed to be public.
- **<return-type>:**
  - the Java type of any output generated by this method.
  - If the method does not produce any returned output, the return type is void.
  - A return type must always be specified.
- **The method-name:**
  - any valid name, by convention starting with a lower-case letter.
- **The <formal-parameters>:**
  - a list of zero or more parameters that the method will take as inputs to do the job that is designed to perform.
  - If there are no inputs, the set of empty round brackets must still be included to denote that this is a method definition.
- **return statement:**
  - if the method produces a value to be returned, i.e. it is not void, then the return statement is used to signify the value that is returned.
  - Executing a return statement at any point during a method invocation causes the method to finish.

METHODS

- An example method definition that takes two formal parameters and produces an output:

```
public int addTwoNumbers(int a, int b)
{
    int c;
    c = a + b;
    return c;
}
```

- To use this method, we would use some calling code (within another method) like this:

```
int d;
d = addTwoNumbers(3,4);
System.out.println(d);
```

- In this example, we invoke the method addTwoNumbers() with two actual parameters, 3 and 4 in this case.

METHODS

- another example of a method that takes two parameters, determines which is larger and displays an appropriate message on the console:

```
public void showLarger(int a, int b){
     if (a > b){
            System.out.println("a is larger");
     }
     else if (b > a){
            System.out.println("b is larger");
     }
     else {
            System.out.println("a is equal to b");
     }
}
```

- Note that as this method is void, there is no return keyword

USING METHODS

1. Design the algorithm.
   - Look at the structure, and which commands are repeated.
   - Decide what are the important overall tasks.

2. Declare (write down) the methods.
   - Arrange statements into groups and give each group a name.

3. Call (run) the methods.
   - The program's main method executes the other methods to perform the overall task.

DESIGN OF AN ALGORITHM…

```
// This program displays a delicious recipe for baking cookies.
public class BakeCookies2 {
  public static void main(String[] args) {

    // Step 1: Make the cake batter.
    System.out.println("Mix the dry ingredients.");
    System.out.println("Cream the butter and sugar.");
    System.out.println("Beat in the eggs.");
    System.out.println("Stir in the dry ingredients.");

    // Step 2a: Bake cookies (first batch).
    System.out.println("Set the oven temperature.");
    System.out.println("Set the timer.");
    System.out.println("Place a batch of cookies into the oven.");
    System.out.println("Allow the cookies to bake.");
```

…DESIGN OF AN ALGORITHM

```
    // Step 2b: Bake cookies (second batch).
    System.out.println("Set the oven temperature.");
    System.out.println("Set the timer.");
    System.out.println("Place a batch of cookies into the oven.");
    System.out.println("Allow the cookies to bake.");

    // Step 3: Decorate the cookies.
    System.out.println("Mix ingredients for frosting.");
    System.out.println("Spread frosting and sprinkles.");
  }
}
```

DECLARING A METHOD

- Gives your method a name so it can be executed
- Syntax:

```
public static void name() {
    statement;
    statement;
    ...
    statement;
}
```

- Example:

```
public static void printWarning() {
   System.out.println("This product causes cancer");
   System.out.println("in lab rats and humans.");
}
```

CALLING A METHOD

- Executes the method's code
- Syntax:

```
name();
```

  - You can call the same method many times if you like.

- Example:

```
printWarning();
```

- Output:

```
This product causes cancer
in lab rats and humans.
```

PROGRAM WITH A METHOD

```
public class RapLyrics {
   public static void main(String[] args) {
      rap(); // Calling (running) the rap method
      System.out.println();
      rap(); // Calling the rap method again
   }
   // This method prints the lyrics to my favorite song.
   public static void rap() {
      System.out.println("İstisnalar kaideyi bozmaz, ");
      System.out.println("Kuru yanında yaş telas yapmaz. ");
   }
}
```

- Output:

```
Istisnalar kaideyi bozmaz,
Kuru yanında yas telas yapmaz.

Istisnalar kaideyi bozmaz,
Kuru yanında yas telas yapmaz.
```

FINAL COOKIE PROGRAM…

```
// This program displays a delicious recipe for baking
cookies.
public class BakeCookies3 {
  public static void main(String[] args) {
    makeBatter();
    bake(); // 1st batch
    bake(); // 2nd batch
    decorate();
  }

  // Step 1: Make the cake batter.
  public static void makeBatter() {
    System.out.println("Mix the dry ingredients.");
    System.out.println("Cream the butter and sugar.");
    System.out.println("Beat in the eggs.");
    System.out.println("Stir in the dry ingredients.");
  }
```

…FINAL COOKIE PROGRAM

```
  // Step 2: Bake a batch of cookies.
  public static void bake() {
    System.out.println("Set the oven temperature.");
    System.out.println("Set the timer.");
    System.out.println("Place a batch of cookies into the
    oven.");
    System.out.println("Allow the cookies to bake.");
  }

  // Step 3: Decorate the cookies.
  public static void decorate() {
    System.out.println("Mix ingredients for frosting.");
    System.out.println("Spread frosting and sprinkles.");
  }
}
```

FINAL COOKIE PROGRAM

```
// This program displays a delicious recipe for baking cookies.
public class BakeCookies3 {
   public static void main(String[] args) {
      makeBatter();
      bake(); // 1st batch
      bake(); // 2nd batch
      decorate();
   }
   // Step 1: Make the cake batter.
   public static void makeBatter() {
      System.out.println("Mix the dry ingredients.");
      System.out.println("Cream the butter and sugar.");
      System.out.println("Beat in the eggs.");
      System.out.println("Stir in the dry ingredients.");
   }
   // Step 2: Bake a batch of cookies.
   public static void bake() {
      System.out.println("Set the oven temperature.");
      System.out.println("Set the timer.");
      System.out.println("Place a batch of cookies into the oven.");
      System.out.println("Allow the cookies to bake.");
   }
   // Step 3: Decorate the cookies.
   public static void decorate() {
      System.out.println("Mix ingredients for frosting.");
      System.out.println("Spread frosting and sprinkles.");
   }
}
```

METHODS CALLING METHODS

```
public class MethodsExample {
      public static void main(String[] args) {
                message1();
                message2();
                System.out.println("Done with main.");
      }
      public static void message1() {
                System.out.println("This is message1.");
      }
      public static void message2() {
                System.out.println("This is message2.");
                message1();
                System.out.println("Done with message2.");
      }
}
```

- Output:

```
This is message1.
This is message2.
This is message1.
Done with message2.
Done with main.
```

CONTROL FLOW

- When a method is called, the program's execution…
  - "jumps" into that method, executing its statements, then
  - "jumps" back to the point where the method was called.

```
public class MethodsExample {
    public static void main(String[] args) {
        message1();

        message2();

        System.out.println("Done with main.");
    }
    ...
}
```

```
public static void message1() {
    System.out.println("This is message1.");
}
```

```
public static void message2() {
    System.out.println("This is message2.");
    message1();

System.out.println("Done with message2.");
}
```

```
public static void message1() {
    System.out.println("This is message1.");
}
```

WHEN TO USE METHODS

- Place statements into a method if:
  - The statements are related structurally, and/or
  - The statements are repeated.

- You should not create methods for:
  - An individual `println` statement.
  - Only blank lines.
    - (Put blank `println`s in `main`.)
  - Unrelated or weakly related statements.
    - (Consider splitting them into two smaller methods.)

Any Questions?