جامعة بيروت العربية
BEIRUT ARAB UNIVERSITY

CMPS 241
Introduction to Programming

Arrays – Part II

Scope

Static Methods

Quick array initialization

type`[]` **name** `=` `{`**value**`,` **value**`,` … **value**`};`

– Example:

```
int[] numbers = {12, 49, -2, 26, 5, 17, -6};
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	12	49	-2	26	5	17	-6

– Useful when you know what the array's elements will be

– The compiler figures out the size by counting the values

Array traversal

- **traversal**: An examination of each element of an array.
- What element are prime numbers in the following array?

```
int[] a = {1, 7, 5, 6, 4, 14, 11};
for (int i = 0; i < a.length; i++) {
    if (isPrime(a[i])) {
        System.out.println(a[i]);
    }
}
```

Output:

7
5
11

Array traversal (Example)

```
// Find the smallest element in an array.
public class Smallest {
    public static void main(String[] args) {
       int[] num = {12, 49, -5, 26, -2, 17, 6}; // array
                                                   initialization
       int min = num[0];
       int minIndex = 0;
        for (int i = 1; i < num.length; i++) {
           if (num[i] < min) {
                 min = num[i];
                 minIndex = i;
             }
         }
         // report results
        System.out.println("The smallest element is: " + min);
       System.out.println("The index of the smallest element is: "
  + minIndex);
    }
}
```

"Array mystery" problem

- What element values are stored in the following array?

```
int[] a = {1, 7, 5, 6, 4, 14, 11};
for (int i = 0; i < a.length - 1; i++)
  {
     if (a[i] > a[i + 1]) {
         a[i + 1] = a[i + 1] * 2;
     }
}
```

index	*0*	*1*	*2*	*3*	*4*	*5*	*6*
value	1	7	10	12	8	14	22

Limitations of arrays

- You cannot resize an existing array:

```
int[] a = new int[4];
a.length = 10;              // error
```

- You cannot compare arrays with == or `equals`:

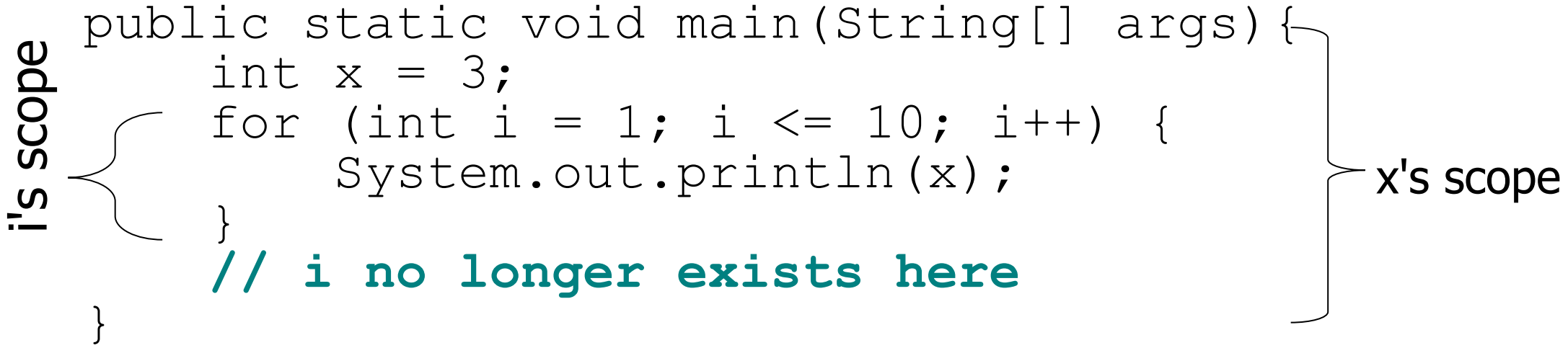
```
int[] a1 = {42, -7, 1, 15};
int[] a2 = {42, -7, 1, 15};
if (a1 == a2) { ... }           // false!
if (a1.equals(a2)) { ... }      // false!
```

- An array does not know how to print itself:

```
int[] a1 = {42, -7, 1, 15};
System.out.println(a1);         // [I@98f8c4]
```

Scope

- **scope**: The part of a program where a variable exists.
  - From its declaration to the end of the **{ }** braces
    - A variable declared in a `for` loop exists only in that loop.
    - A variable declared in a method exists only in that method.



Scope implications

- Variables without overlapping scope can have same name.

```
for (int i = 1; i <= 100; i++) {
    System.out.print("/");
}
for (int i = 1; i <= 100; i++) {   // OK
    System.out.print("\\");
}
int i = 5;                         // OK: outside of loop's scope
```

- A variable can't be declared twice or used out of its scope.

```
for (int i = 1; i <= 100 * line; i++) {
    int i = 2;                     // ERROR: overlapping scope
    System.out.print("/");
}
i = 4;                             // ERROR: outside scope
```

Static methods

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 oven temperature.
  - 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 Add frosting and sprinkles.
  - Mix the ingredients for the frosting.
  - Spread frosting and sprinkles onto the cookies.

  ...

Removing redundancy

- A well-structured algorithm can describe repeated tasks with minimum redundancy.

1 Make the cookie batter.
- Mix the dry ingredients.
- ...

2a Bake the cookies (first batch).
- Set the oven temperature.
- Set the timer.
- ...

2b Bake the cookies (second batch).

3 Decorate the cookies.
- ...

A program with redundancy

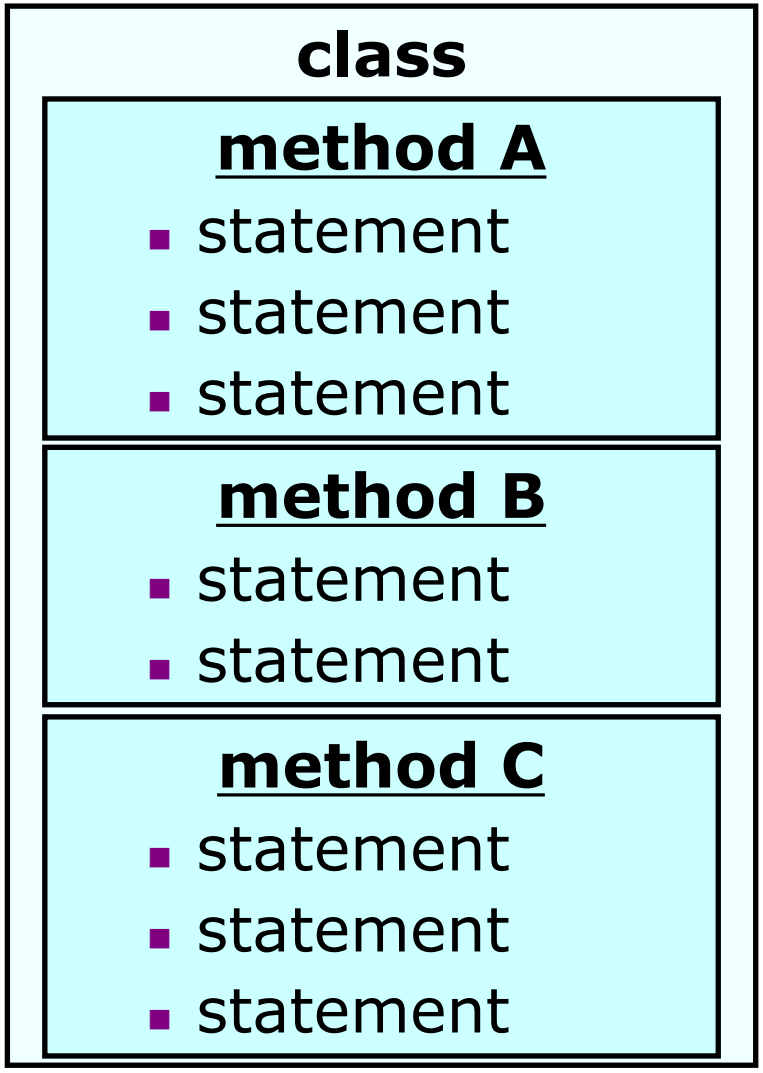
```
public class BakeCookies {
    public static void main(String[] args) {
        System.out.println("Mix the dry ingredients.");
        System.out.println("Cream the butter and sugar.");
        System.out.println("Beat in the eggs.");
        System.out.println("Stir in the dry ingredients.");
        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.");
    }
}
```

Static methods

- **static 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 **static method** is like adding a new command to Java.



Using static 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.

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 for as many times as you like.

- Example:

```
printWarning();
```

  – Output:

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

Cookie program

```
//This program displays a delicious recipe for baking cookies.
public class BakeCookies {
 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.");
 }
}
```

Program with static method

```
public class FreshPrince {
    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("Now this is the story all about how");
        System.out.println("My life got flipped turned upside-down");
    }
}
```

Output:

```
Now this is the story all about how
My life got flipped turned upside-down

Now this is the story all about how
My life got flipped turned upside-down
```

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
    }

    ...
}
```

call

return

```
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

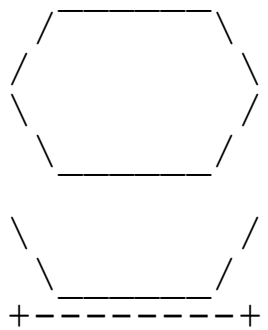
- Place statements into a static method if:
  - The statements are related structurally, and/or
  - The statements are repeated.

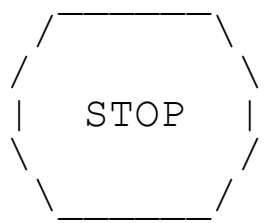
- You should not create static methods for:
  - An individual `println` statement.
  - Only blank lines (put blank `println` statements in main).
  - Unrelated or weakly related statements
    (consider splitting them into two smaller methods).

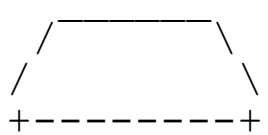
Drawing complex figures with static methods

Static methods question

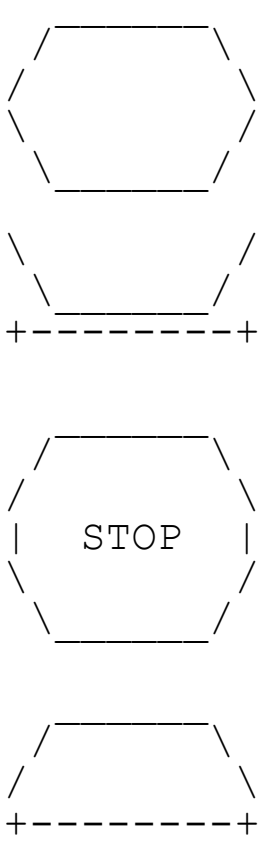
- Write a program to print these figures using methods.







Development strategy



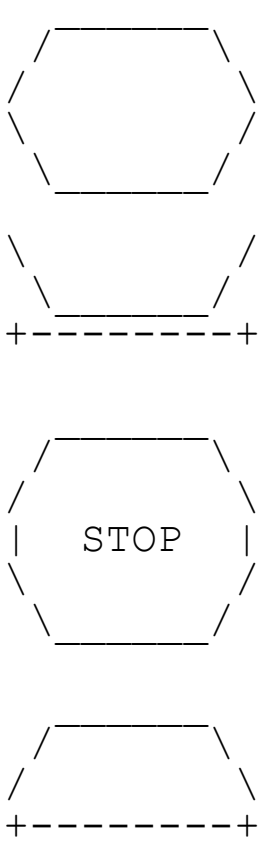
First version (unstructured):

- Create an empty program and `main` method.
- Copy the expected output into it, surrounding each line with `System.out.println` syntax.
- Run it to verify the output.

Program version 1

```
public class Figures1 {
    public static void main(String[] args) {
        System.out.println("   ______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("\\         /");
        System.out.println(" \\______/");
        System.out.println();
        System.out.println("\\         /");
        System.out.println(" \\______/");
        System.out.println("+--------+");
        System.out.println();
        System.out.println("   ______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("|  STOP   |");
        System.out.println("\\         /");
        System.out.println(" \\______/");
        System.out.println();
        System.out.println("   ______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("+--------+");
    }
}
```

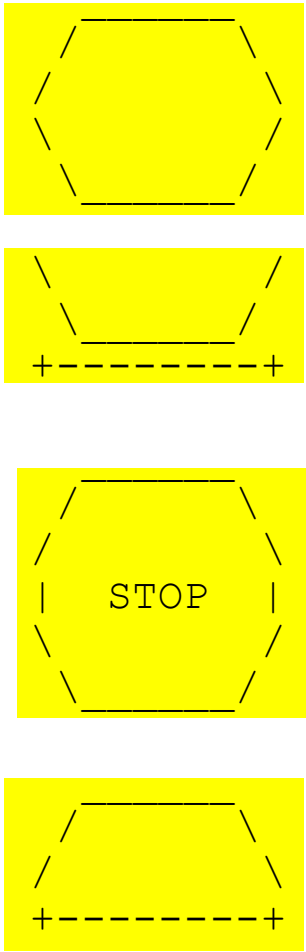
Development strategy 2



Second version (structured, with redundancy):

- Identify the structure of the output.
- Divide the `main` method into static methods based on this structure.

Output structure



The structure of the output:

- initial "egg" figure
- second "teacup" figure
- third "stop sign" figure
- fourth "hat" figure

This structure can be represented by methods:

- `egg`
- `teaCup`
- `stopSign`
- `hat`

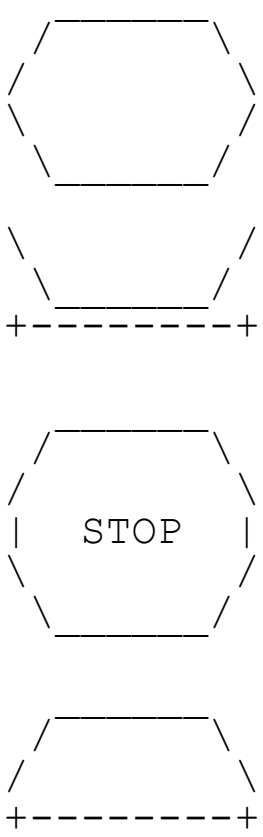
Program version 2

```
public class Figures2 {
    public static void main(String[] args) {
        egg();
        teaCup();
        stopSign();
        hat();
    }

    public static void egg() {
        System.out.println("  ______");
        System.out.println(" /      \\");
        System.out.println("/        \\");
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println();
    }

    public static void teaCup() {
        System.out.println("\\        /");
        System.out.println(" \\______/");
        System.out.println("+--------+");
        System.out.println();
    }
    ...
```

Program version 2, cont'd.

```
  ...
  public static void stopSign() {
        System.out.println("   _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("|  STOP   |");
        System.out.println("\\         /");
        System.out.println(" \\_______/");
        System.out.println();
    }

    public static void hat() {
        System.out.println("   _______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
        System.out.println("+---------+");
    }
}
```

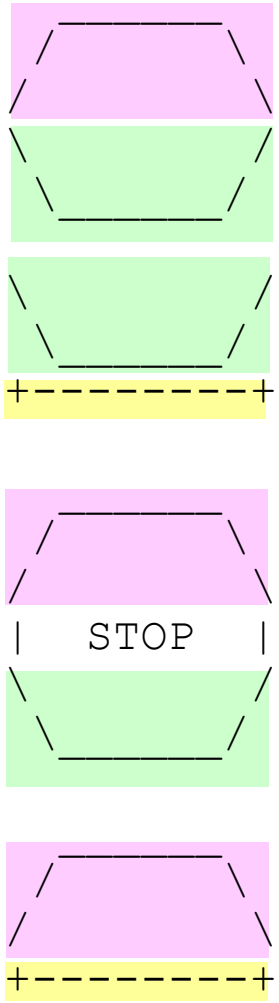
Development strategy 3



Third version (structured, without redundancy):

- Identify redundancy in the output, and create methods to eliminate as much as possible.
- Add comments to the program.

Output redundancy



The redundancy in the output:

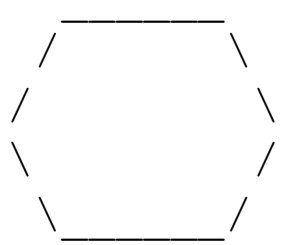
- **egg top**: reused on stop **sign**, **hat**
- **egg bottom**: reused on **teacup**, **stop sign**
- **divider line**: used on **teacup**, **hat**

This redundancy can be fixed by methods:

- `eggTop`
- `eggBottom`
- `line`

Program Version 3

```
//Prints several figures, with methods for structure and redundancy.
public class Figures3 {
    public static void main(String[] args) {
        egg();
        teaCup();
        stopSign();
        hat();
    }

    // Draws the top half of an an egg figure.
    public static void eggTop() {
        System.out.println("   ______");
        System.out.println(" /       \\");
        System.out.println("/         \\");
    }

    // Draws the bottom half of an egg figure.
    public static void eggBottom() {
        System.out.println("\\         /");
        System.out.println(" \\_____/");
    }

    // Draws a complete egg figure.
    public static void egg() {
        eggTop();
        eggBottom();
        System.out.println();
    }
    ...
```



Program version 3, cont'd.

```
    // Draws a teacup figure.
    public static void teaCup() {
        eggBottom();
        line();
        System.out.println();
    }

    // Draws a stop sign figure.
    public static void stopSign() {
        eggTop();
        System.out.println("|  STOP  |");
        eggBottom();
        System.out.println();
    }

    // Draws a figure that looks sort of like a hat.
    public static void hat() {
        eggTop();
        line();
    }

    // Draws a line of dashes.
    public static void line() {
        System.out.println("+--------+");
    }
}
```