Figures.java

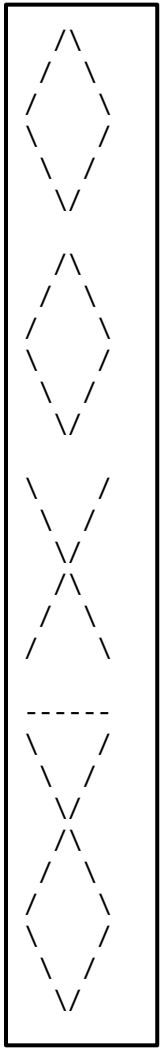
Program Breakdown
(Static Methods)

Problem Statement

Write a program called Figures.java that prints the output on the right to the console.

Structure the code so there are no non-blank System.out.println() calls in main.

Reduce whole line redundancy as much as possible!

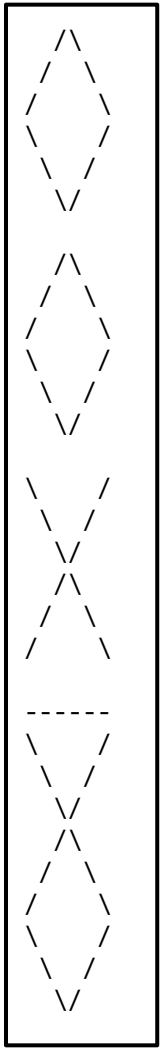


Problem Statement

Write a program called Figures.java that prints the output on the right to the console.

Structure the code so there are no non-blank System.out.println() calls in main.

Reduce whole line redundancy as much as possible!



What we need

Way to write a program that prints output

- ❏ class with main method and System.out.println() calls

Way to represent output text in Java

- ❏ String literals for each line of output we want to produce

Way to move code out of main but still have it executed

- ❏ ????

Way to reduce redundancy

- ❏ ????

Development Strategy

1. Unstructured: Print all output in main
2. Structured: Use static methods to produce the output without non-blank System.out.println statements in main
3. Reduce redundancy: Use static methods to reduce redundancy in the code

Escape Sequences

NOTE: You should not use escape sequences on Take-Home Assessment 1 (nor will they be needed)

Motivation:

- Some characters we are unable to represent within the String on their own (e.g. quotation mark character " is being used to indicate the start and end of a String of characters)
- Use a special sequence of characters (escape sequence) to represent these outliers in the String instead

Examples:

- Quotation mark is represented by \"
- Backslash is represented by \\
  - Why do we need an escape sequence for backslash?

Anywhere in we want to put a backslash character (\) in a String, we have to use the escape sequence instead (\\)

```
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();
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("/    \\");
System.out.println("\\    /");
System.out.println(" \\  /");
System.out.println("  \\/");
```

Development Strategy

1. ~~Unstructured: Print all output in main~~
2. Structured: Use static methods to produce the output without non-blank System.out.println statements in main
3. Reduce redundancy: Use static methods to reduce redundancy in the code

Static Methods

```
public static void methodName() {


}
```

Specifying your method:

- You can name a method anything you want as long as:
  - It contains only letters, numbers, underscores, and dollar sign symbols
  - Must begin with a letter
- By convention, we use camelCasing
  - First word is lowercase
  - Every next word is capitalized
- Should be descriptive of the method's task

****Keywords:**

- These words (public static void) indicate (declare) to Java that you are writing a method that contains statements of executable code
- A method is a subroutine, a part of the overall procedure of the program that has been labeled
- Main is a special method that begins executing when you run the program

Scope:

- Everything between the curly braces is part of (within the scope of) the main method
- This is where any number of executable statements of code go
- Java will execute the statements in the method in order from top to bottom

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello from main");
        printGreeting();
        System.out.println("Hello again from main");
        printGreeting();
    }

    public static void printGreeting() {
        System.out.println("Hello from printGreeting");
    }
}
```

Output:

Control flow

- We hit run and Java goes to the main method and starts executing statements from top to bottom

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello from main");
        printGreeting();
        System.out.println("Hello again from main");
        printGreeting();
    }

    public static void printGreeting() {
        System.out.println("Hello from printGreeting");
    }
}
```

Output:

```
Hello from main
```

Control flow

- We hit run and Java goes to the main method and starts executing statements from top to bottom
- Java reaches the first println statement in main and executes it

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello from main");
        printGreeting();
        System.out.println("Hello again from main");
        printGreeting();
    }

    public static void printGreeting() {
        System.out.println("Hello from printGreeting");
    }
}
```

Output:

```
Hello from main
```

Control flow

- We hit run and Java goes to the main method and starts executing statements from top to bottom
- Java reaches the first println statement in main and executes it
- Java reaches the first call to printGreeting, so it jumps down to the printGreeting method and starts executing statements from top to bottom

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello from main");
        printGreeting();
        System.out.println("Hello again from main");
        printGreeting();
    }

    public static void printGreeting() {
        System.out.println("Hello from printGreeting");
    }
}
```

Output:

```
Hello from main
Hello from printGreeting
```

Control flow

- We hit run and Java goes to the main method and starts executing statements from top to bottom
- Java reaches the first println statement in main and executes it
- Java reaches the first call to printGreeting, so it jumps down to the printGreeting method and starts executing statements from top to bottom
- Java reaches the first println statement in printGreeting and executes it

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello from main");
        printGreeting();
        System.out.println("Hello again from main");
        printGreeting();
    }

    public static void printGreeting() {
        System.out.println("Hello from printGreeting");
    }
}
```

Output:

```
Hello from main
Hello from printGreeting
Hello again from main
```

Control flow

- We hit run and Java goes to the main method and starts executing statements from top to bottom
- Java reaches the first println statement in main and executes it
- Java reaches the first call to printGreeting, so it jumps down to the printGreeting method and starts executing statements from top to bottom
- Java reaches the first println statement in printGreeting and executes it
- Java goes back up to main and continues executing statements, so this next println is executed

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello from main");
        printGreeting();
        System.out.println("Hello again from main");
        printGreeting();
    }

    public static void printGreeting() {
        System.out.println("Hello from printGreeting");
    }
}
```

Output:

```
Hello from main
Hello from printGreeting
Hello again from main
```

Control flow

- We hit run and Java goes to the main method and starts executing statements from top to bottom
- Java reaches the first println statement in main and executes it
- Java reaches a call to printGreeting, so it jumps down to the printGreeting method and starts executing statements from top to bottom
- Java reaches the first println statement in printGreeting and executes it
- Java goes back up to main and continues executing statements, so this next println is executed
- Java reaches a call to printGreeting, so it jumps down to the printGreeting method and starts executing statements from top to bottom

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello from main");
        printGreeting();
        System.out.println("Hello again from main");
        printGreeting();
    }

    public static void printGreeting() {
        System.out.println("Hello from printGreeting");
    }
}
```

Output:

```
Hello from main
Hello from printGreeting
Hello again from main
Hello from printGreeting
```

Takeaway:
Static methods allow segments of code (subtasks) to be moved out of the main method's statements but still get executed!!!

Control flow

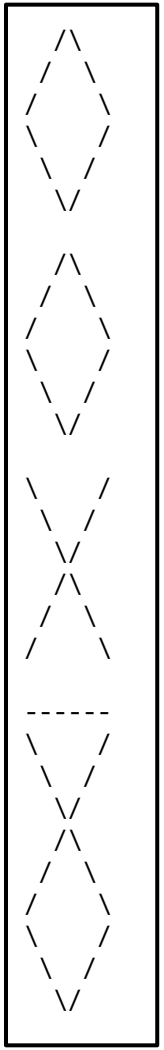
- We hit run and Java goes to the main method and starts executing statements from top to bottom
- Java reaches the first println statement in main and executes it
- Java reaches a call to printGreeting, so it jumps down to the printGreeting method and starts executing statements from top to bottom
- Java reaches the println statement in printGreeting and executes it
- Java goes back up to main and continues executing statements, so this next println is executed
- Java reaches a call to printGreeting, so it jumps down to the printGreeting method and starts executing statements from top to bottom
- Java reaches the println statement in printGreeting and executes it

Problem Statement

Write a program called Figures.java that prints the output on the right to the console.

Structure the code so there are no non-blank System.out.println() calls in main.

Reduce whole line redundancy as much as possible!



What we need

Way to write a program that prints output

- ❏ class with main method and System.out.println() calls

Way to represent output text in Java

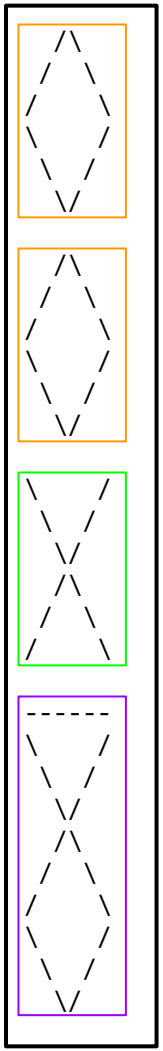
- ❏ String literals for each line of output we want to produce

Way to move code out of main but still have it executed

- ❏ Static methods

Way to reduce redundancy

- ❏ Static methods



```
  /\
 /  \
/    \
\    /
 \  /
  \/
```

```
  /\
 /  \
/    \
\    /
 \  /
  \/
```

```
\    /
 \  /
  \/
  /\
 /  \
/    \
```

```
------
\    /
 \  /
  \/
  /\
 /  \
/    \
\    /
 \  /
  \/
```

```
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();
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("/    \\");
System.out.println("\\    /");
System.out.println(" \\  /");
System.out.println("  \\/");
```

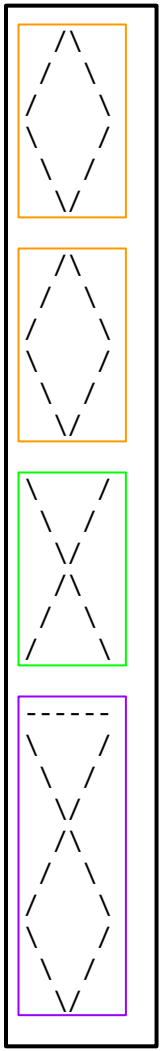
Structure

Motivation:
We can identify subtasks in our program, even visually from the output

- Diamond figure
- Diamond figure
- X figure
- Tie

Since main is the method that Java is going to start executing and return to executing, it is in control of the whole program

- Main should be a concise summary of the program
- Like a table of contents showing the subtasks of ***what*** the program will do ***without showing how*** the code for the subtasks is written



```
  /\
 /  \
/    \
\    /
 \  /
  \/
```

```
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();
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("/    \\");
System.out.println("\\    /");
System.out.println(" \\  /");
System.out.println("  \\/");
```

```
public static void printDiamond() {
    System.out.println("  /\\");
    System.out.println(" /  \\");
    System.out.println("/    \\");
    System.out.println("\\    /");
    System.out.println(" \\  /");
    System.out.println("  \\/");
}

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

public static void printTie() {
    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("  \\/");
}
```

```
public static void main(String[] args) {
    printDiamond();
    System.out.println();
    printDiamond();
    System.out.println();
    printX();
    System.out.println();
    printTie();
}
```

Structural components:

- Our main method is a concise summary of what we visually see in the program and there are no non-blank println's!
- We only need one definition of the printDiamond method and we can call it twice to achieve the same output twice without rewriting the code!

Development Strategy

1. ~~Unstructured: Print all output in main~~
2. ~~Structured: Use static methods to produce the output without non-blank System.out.println statements in main~~
3. Reduce redundancy: Use static methods to reduce redundancy in the code

```
public static void printDiamond() {
    System.out.println("  /\\");
    System.out.println(" /  \\");
    System.out.println("/    \\");
    System.out.println("\\    /");
    System.out.println(" \\  /");
    System.out.println("  \\/");
}
```

```
public static void printMountain() {
    System.out.println("  /\\");
    System.out.println(" /  \\");
    System.out.println("/    \\");
}
```

```
public static void printDiamond() {
    printMountain();
    printValley();
}
```

```
public static void printX() {
    System.out.println("\\    /");
    System.out.println(" \\  /");
    System.out.println("  \\/");
    System.out.println("  /\\");
    System.out.println(" /  \\");
    System.out.println("/    \\");
}
```

```
public static void printValley() {
    System.out.println("\\    /");
    System.out.println(" \\  /");
    System.out.println("  \\/");
}
```

```
public static void printX() {
    printValley();
    printMountain();
}
```

Redundancy Definition for now: 2 or more consecutive lines of code that appear in 2 or more places.

Motivation:

- We can identify repeated subtasks within our code and create new methods for the code:
  - The red boxes
  - The blue boxes
- Using methods to wrap repeated code under one subtask reduces redundancy!

```
public static void printTie() {
    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("  \\/");
}
```

```
public static void printMountain() {
    System.out.println("  /\\");
    System.out.println(" /  \\");
    System.out.println("/    \\");
}
```

```
public static void printValley() {
    System.out.println("\\    /");
    System.out.println(" \\  /");
    System.out.println("  \\/");
}
```

```
public static void printDiamond() {
    printMountain();
    printValley();
}
```

```
public static void printX() {
    printValley();
    printMountain();
}
```

```
public static void printTie() {
    System.out.println("------");
    printX();
    printValley();
}
```

Redundancy

Definition for now: 2 or more consecutive lines of code that appear in 2 or more places.

Motivation:

- We can identify repeated subtasks for which we already have methods:
  - The blue box (printValley)
  - The green box (printX)
- We don't need to rewrite ANY code that already has a method that executes the code

```
public static void main(String[] args) {
    printDiamond();
    System.out.println();
    printDiamond();
    System.out.println();
    printX();
    System.out.println();
    printTie();
}
```

Main method:

- Directs what happens in the program without doing the actual work of printing text out
- Is a concise summary of our program
- Contains no non-blank println statements

```
public static void printDiamond() {
    printMountain();
    printValley();
}
```

```
public static void printX() {
    printValley();
    printMountain();
}
```

```
public static void printTie() {
    System.out.println("******");
    printX();
    printValley();
}
```

Methods for structure:

- These methods show the primary components of the program (the different figures)
- They are used to structure main
- printDiamond reduces redundancy in main
- printX reduces redundancy in printTie

```
public static void printValley() {
    System.out.println("\\    /");
    System.out.println(" \\  /");
    System.out.println("  \\/");
}
```

```
public static void printMountain() {
    System.out.println("  /\\");
    System.out.println(" /  \\");
    System.out.println("/    \\");
}
```

Methods for redundancy:

- These methods reduce redundancy by identifying subtasks within the structure methods
- They are not called from main but the code is executed because they are called from other methods (printDiamond, printX, printTie) which are called from main

Trivial Methods

There are cases where writing a method to wrap a statement of code does not actually help structure or reducing redundancy. This is usually (but not always) the case when you have a method that could be replaced with a single statement of code.

```
public static void printTie() {
    System.out.println("------");
    printX();
    printValley();
}
```

This line does not need a method to wrap it!! One println does not meet our definition of 2 or more consecutive lines of code in 2 or more places

```
public static void printTie() {
    printLine();
    printX();
    printValley();
}

public static void printLine() {
    System.out.println("------");
}
```

All we've done is replaced one statement of code

`System.out.println("------");`

with one method call

`printLine();`

This is a trivial method!! It doesn't add to structure and it doesn't reduce redundancy

```
public static void printDiamond() {
    System.out.println("  /\\");
    System.out.println(" /  \\");
    System.out.println("/    \\");
    System.out.println("\\    /");
    System.out.println(" \\  /");
    System.out.println("  \\/");
}

public static void printX() {
    System.out.println("\\    /");
    System.out.println(" \\  /");
    System.out.println("  \\/");
    System.out.println("  /\\");
    System.out.println(" /  \\");
    System.out.println("/    \\");
}
```

```
public static void printDiamond() {
    mountainLine1();
    mountainLine2();
    mountainLine3();
    valleyLine1();
    valleyLine2();
    valleyLine3();
}

public static void printX() {
    mountainLine1();
    mountainLine2();
    mountainLine3();
    valleyLine1();
    valleyLine2();
    valleyLine3();
}
```

```
public static void mountainLine1() {
    System.out.println("  /\\");
}
public static void mountainLine2() {
    System.out.println(" /  \\");
}
public static void mountainLine3() {
    System.out.println("/    \\");
}
public static void valleyLine1() {
    System.out.println("\\    /");
}
public static void valleyLine2() {
    System.out.println(" \\  /");
}
public static void valleyLine3() {
    System.out.println("  \\/");
}
```

Trivial methods cannot reduce redundancy!

Notice that even though none of the System.out.println statements are repeated, the code STILL has redundancy (2 or more consecutive lines of code in two or more places)….it's just static method calls instead of System.out.println statements.

The mountain and valley line methods are all trivial, since they do not reduce redundancy or add to structure.