# Ch. 6

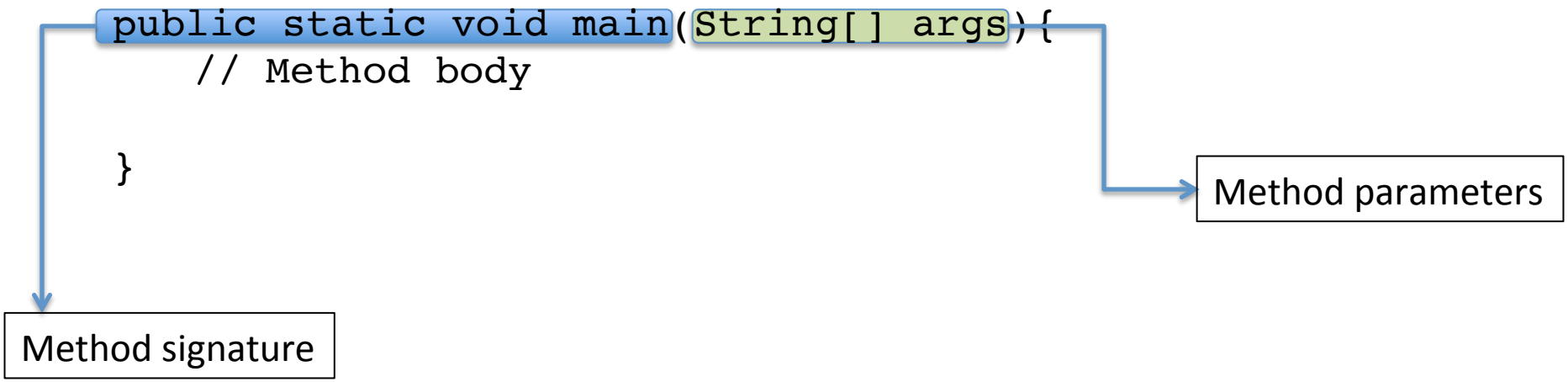
User-Defined Methods

# Functional Abstraction

- Functional − regarding functions/methods
- Abstraction − solving a problem in a creative way
- Stepwise refinement − breaking down large problems into small problems
- The flow of control should be as simple as possible

# Let's review: main

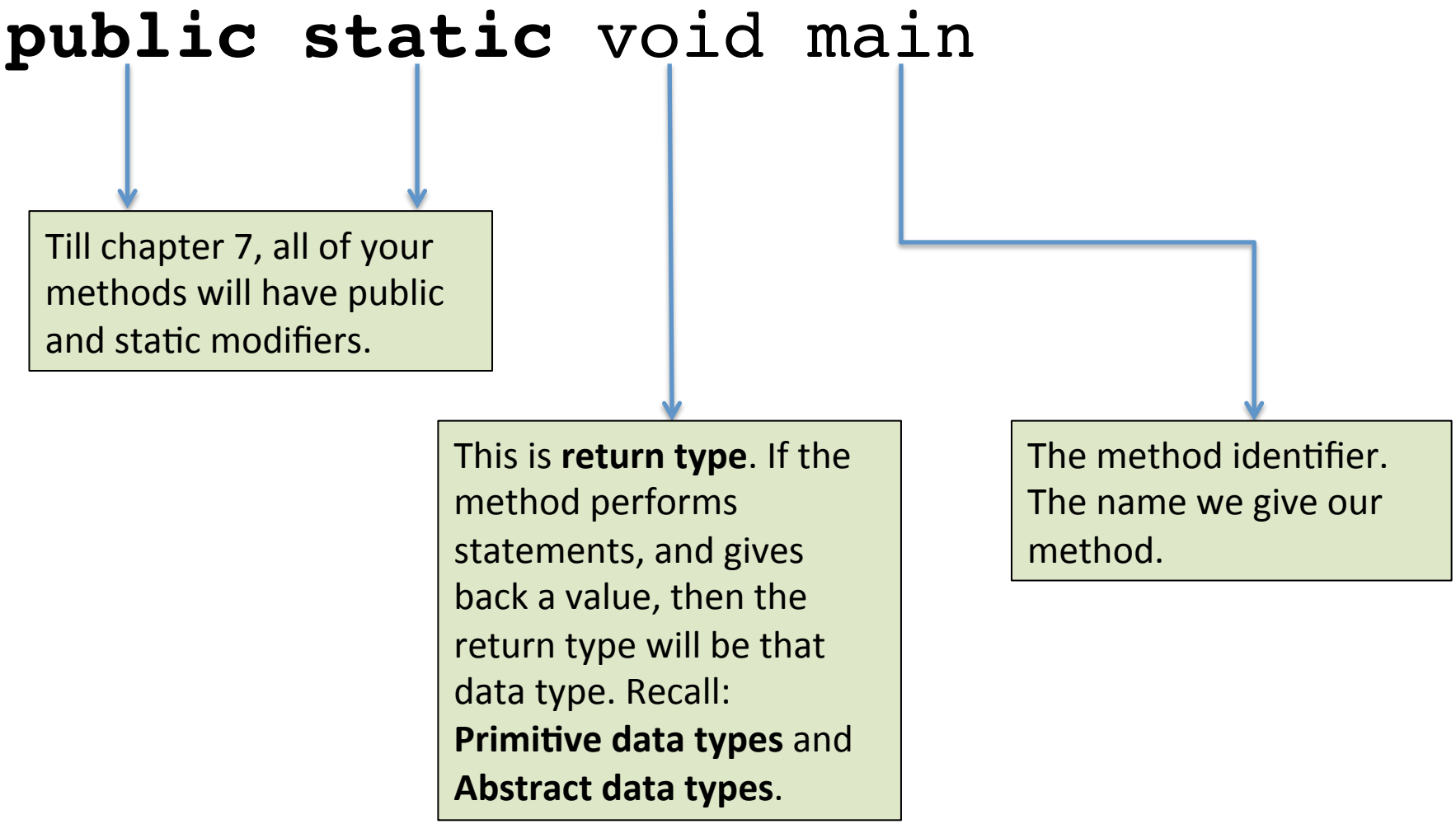
main method is a user defined method we have used in every program!



# 2 Aspects

- Calling methods
- Defining methods

# Method signature



# Method Parameters

- This will be a list of variables that will be used to pass information to the method.
- You don't have to give parameters, but if you do, separate them with commas.

Ex:

```
public static void addInts(int arg1, int arg2){
   System.out.println(arg1 + arg2);
}
```

```
// This program demonstrates a very simple method, drawGrid,
// that draws a 5 by 5 grid of asterisks. This method is called
// from main.

public class Grid{
    public static void main(String[] args){
        drawGrid();
    }

    // method to draw a 5x5 grid. Accepts no parameters and
    // returns nothing.
    public static void drawGrid(){
        for(int i = 0; i < 5; i++){
            for(int j = 0; j < 5; j++){
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

```
import java.util.Scanner;
public class DrawLines{
    public static void main(String[] args){
        System.out.println("Enter # of asterisks on line.");
        int x = keyboard.nextInt();
        while(x != 0){
            drawLines(x);
            System.out.println("Enter # of asterisks on
                                line.");
            x = keyboard.nextInt();
        }
    }
    // method to specified asterisks on one line
    public static void drawLines(int x){
        for(int i = 0; i < x; i++){
            System.out.print("*");
        }
        System.out.println();
    }
}
```

```
import java.util.Scanner;
public class DrawLines{
    public static void main(String[] args){
        System.out.println("Enter # of asterisks for grid.");
        int x = keyboard.nextInt();
        while(x != 0){
            for(int i = 1; i <= x; i++){
                drawLines(x);
            }
            System.out.println("Enter # of asterisks for grid.");
            x = keyboard.nextInt();
        }
    }
    // method to specified asterisks on one line
    public static void drawLines(int x){
        for(int i = 0; i < x; i++){
            System.out.print("*");
        }
        System.out.println();
    }
}
```

```
import java.util.Scanner;
public class DrawLines{
    public static void main(String[] args){
        System.out.println("Enter #");
        int x = keyboard.nextInt();
        while(x != 0){
            for(int i = 0; i < x; i++){
                drawLines(__________);
            }
            System.out.println("Enter # of asterisks for grid.");
            x = keyboard.nextInt();
        }
    }
    // method to specified asterisks on one line
    public static void drawLines(int x){
        for(int i = 0; i < x; i++){
            System.out.print("*");
        }
        System.out.println();
    }
}
```

Assuming the user enters 5, what goes in the blank to produce the following:

```
*****
****
***
**
*
```

```
import java.util.Scanner;
public class DrawLines{
    public static void main(String[] args){
        System.out.println("Enter #");
        int x = keyboard.nextInt();
    }

    // method to specified asterisks on one line
    public static void drawLines(int x){
        for(int i = 0; i < x; i++){
            System.out.print("*");
        }
        System.out.println();
    }
}
```

What is printed by this program?

# Arguments Vs. Parameters

- An argument is passed to the method when the method is called.
  - Ex: `System.out.println("Program output!");`
- A parameter is expected input coming to the method
  - Ex: `public static void drawLines(int x)`

# `static` keyword

- As mentioned earlier, all of your methods from now till week 6 (chapter 7) will be static
- Since main is a static method, we can only call static methods from a static method.

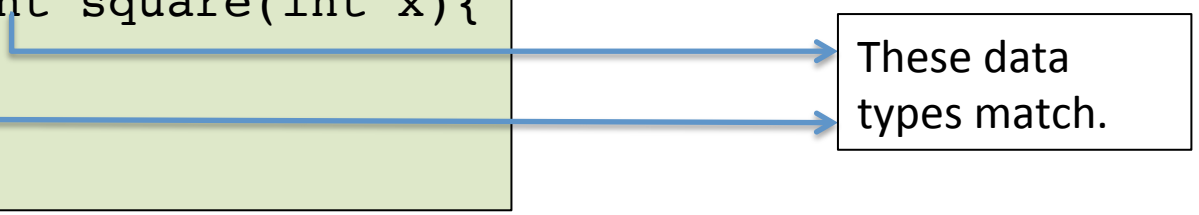
# `return` keyword

- Once the method executes `return`, the method is left immediately
- If our return type is `void`, then our return statement looks like:

```
return;
```

- If our return type is not void, then the data we return must match the return type

```
public static int square(int x){
    ...
    return x*x;
}
```



```
import java.util.Scanner;
public class Minimum{
    public static void main(String[] args){
        Scanner keyboard = new Scanner(System.in);
        int x, y, z;
        System.out.println("Enter first num");
        x = keyboard.nextInt();
        System.out.println("Enter first num");
        y = keyboard.nextInt();
        z = minimum(x, y);
        System.out.println("The min of " + x + " and " + y +
                           " is: " + z);
    }
    // method to calculate minimum of 2 integers
    public static int minimum(int x, int y){
        if(x < y)
            return x;
        return y;
    }
}
```

```
import java.util.Scanner;
public class Mystery2{
    public static void main(String[] args){
        Scanner keyboard = new Scanner(System.in);
        int x;
        mystery2(x);
    }
    // method to print some things to the user
    public static _______ mystery2(int x){
        for(int i = 0; i < x; i++){
            System.out.print("#");
        }
        System.out.println();
    }
}
```

What goes in the blank so this program will compile?

```
import java.util.Scanner;
public class Mystery{
    public static void main(String[] args){
        Scanner keyboard = new Scanner(System.in);
        int x;
        mystery(x);
    }
    // method to compute a simple calculation
    public static _______ mystery(int x){
        double y = Math.sqrt(x);
        return y;
    }
}
```

What goes in the blank so this program will compile?

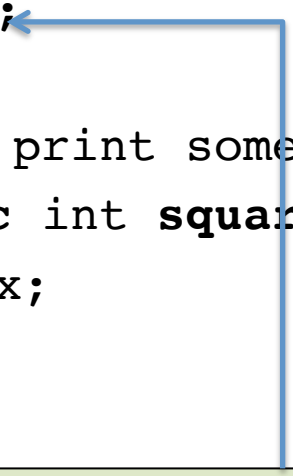
```
public class Mystery{
    public static void main(String[] args){
        ...
    }

    public static _______ mystery(){
        ...
        return data;
    }
}
```

**data** and **blank** must match in data types!

# Why use methods?

- If you find yourself repeating several lines of code over and over, you should probably be using a method.
- Creating methods can also add to readability
- Problems can be easier to tackle when broken down (recall functional abstraction)

```
// A common mistake
import java.util.Scanner;
public class Square{
    public static void main(String[] args){
        Scanner keyboard = new Scanner(System.in);
        int z;
        System.out.println("Enter number to square");
        z = keyboard.nextInt();
        square(z);
    }
    // method to print some things to the user
    public static int square(int x){
        return x*x;
    }
}
```



Simply calling this method is <u>not</u> useful. We need to do something with the value being returned.

```
// Another common mistake
import java.util.Scanner;
public class Mult{
    public static void main(String[] args){
        Scanner keyboard = new Scanner(System.in);
        int x = 5, y = 10;
        System.out.println(x + " multiplied by " + y + " is: " +
                                mult());
    }
    // method to print some things to the user
    public static int mult(){
        return x*y;
    }
}
```

There is a scope issue in this program

```
// Complete the method countChar below such that the method
// counts how many of the specified char are in the string str
// being passed, and returns that value
import java.util.Scanner;
public class CountChar{
    public static void main(String[] args){
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter the String.");
        String str = keyboard.next();
        System.out.println("Enter the char.");
        char ch = keyboard.next().charAt(0);
        System.out.println(countChar(str, ch));
    }
    // method to count # of specified chars
    public static int countChar(String s, char c){
        // Your code goes here

        return ______;
    }
}
```