Methods in Java

Review

What is a Method?

A method is a program within a program, which is also called a sub-program. You may also hear terms such as functions and procedures, which are the same as methods in Java.

We have already made use of the most important method of all, the main method. The main method starts all of our programs.

There are also utility methods, such as those for input, output, and specialized math operations.

Why Use a Method?

1. Organization: Methods allow you to group commands into a task with a meaningful name that summarizes its purpose.

2. Efficiency: Some tasks are repeated many times within a single program. By using a method, these tasks can often be performed with a single line, which calls the method.

3. Maintenance: It is easier to change a single method than change code in multiple locations throughout your program.

Java Structure with Methods
(methods can be declared before main)

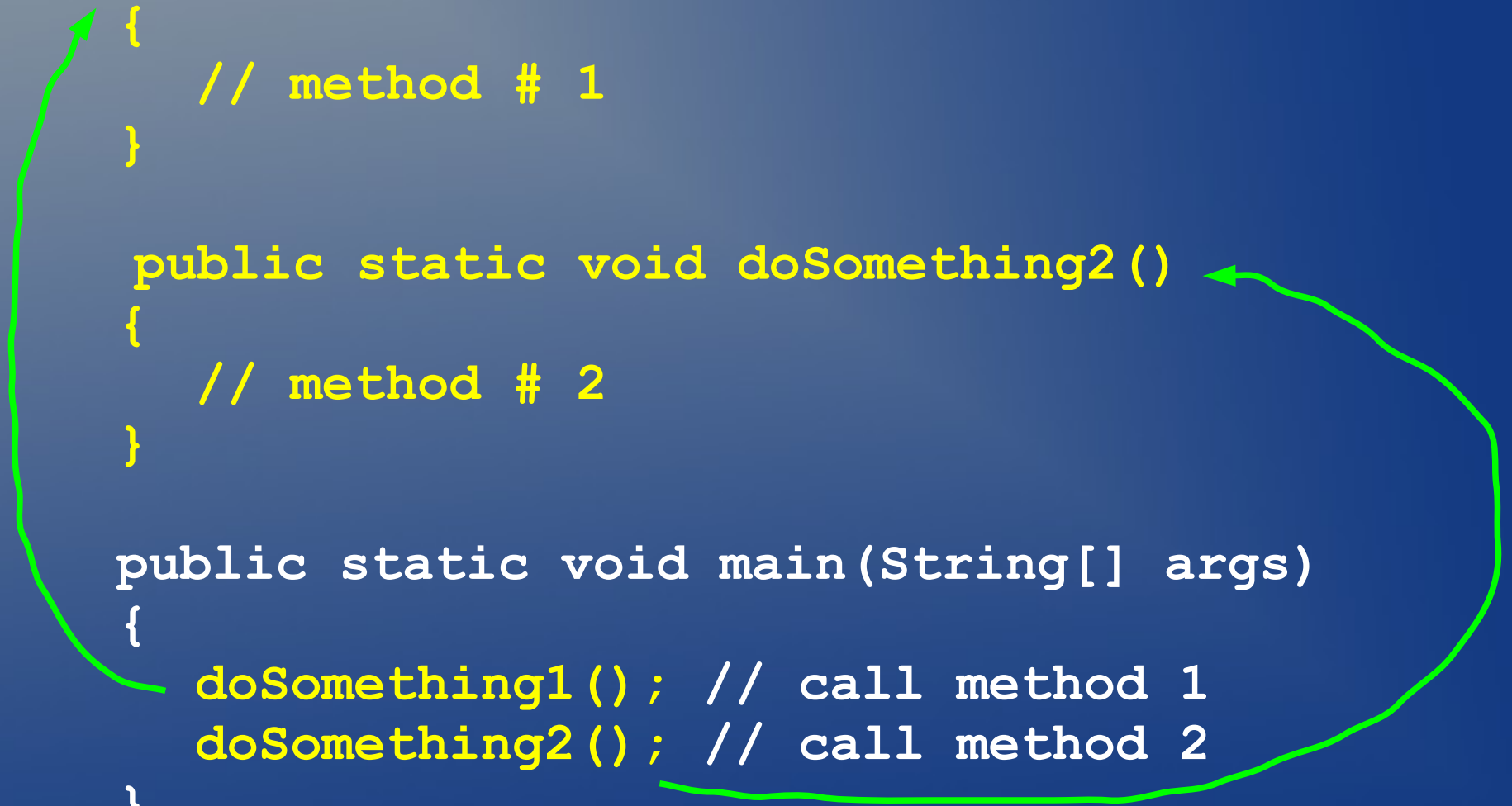
```
class JavaProgram
{
   public static void doSomething()
   {
      // useful instructions here!
   }

   public static void main(String[] args)
   {
      doSomething(); // call method
   }
}
```

Java Structure with Methods
(methods can be declared after main)

```
class JavaProgram
{
   public static void main(String[] args)
   {
      doSomething(); // call method
   }

   public static void doSomething()
   {
      // useful instructions here!
   }
}
```

Multiple Methods in Java

```
class JavaProgram
{
   public static void doSomething1()
   {
      // method # 1
   }

   public static void doSomething2()
   {
      // method # 2
   }

   public static void main(String[] args)
   {
      doSomething1(); // call method 1
      doSomething2(); // call method 2
   }
}
```



Methods Can Call Other Methods

```
class JavaProgram
{
    public static void doSomething1()
    {
        doSomething2(); // call method 2
    }

    public static void doSomething2()
    {
        // method # 2
    }

    public static void main(String[] args)
    {
        doSomething1(); // call method 1
    }
}
```

Remember to always start with main method!

Parameters – Additional Information

When calling, or invoking, a method, we often wish to provide some additional information.

This allows our methods to be more general, which means they can be applied to more situations.

Few Options Without Parameters

```
public static void print10X()
{
   for (int i = 1; i <= 10; i++)
   {
      System.out.print("X");
   }
}

public static void print10Y()
{
   for (int i = 1; i <= 10; i++)
   {
      System.out.print("Y");
   }
}
```

More Options With Parameters

Parameters allow methods to become more generic, and thus more useful and versatile. Whenever possible, avoid having fixed, or predefined, values in a method.

```
public static void print10Chars(char ch)
{
   for (int i = 1; i <= 10; i++)
   {
      System.out.print(ch);
   }
   System.out.println(); // new line at end
}
```

Even More Options With Parameters

A truly generic method may take more effort to create, but it will also have a real chance of being useful in future programs. Code reuse is one of the goals of a good method.

```
public static void printNChars(int n, char ch)
{
   for (int i = 1; i <= n; i++)
   {
      System.out.print(ch);
   }
  System.out.println(); // new line at end
}
```

Parameter Data Protection

Many parameters are one of the basic data types (e.g., int, char, boolean, double).

When passing such data, it is protected by making a copy for the methods. In other words, the method does not get the original – it gets a duplicate.

If the method changes the data, it does not affect the original data.

Warning: Does not apply to more complex data types, such as arrays and Strings.

Parameter Data Protection

```
public static void main (String [] args)
{
   int x = 5;
   doSomething(x);
   System.out.println(x); // x is unchanged
}

public static void doSomething(int a)
{
   a = a + 1;
   System.out.println(a);
}
```

Parameter Data Protection

```
public static void main (String [] args)
{
   int x = 5;
   doSomething(x);
   System.out.println(x);
}

public static void doSomething(int a)
{
   a = a + 1;
   System.out.println(a);
}
```

'a' does not exist here

'x' does not exist here, just a copy of the data

Output:

6

5

Java Structure with Methods
the keyword: void

```
public static void doSomething()
{
   // useful instructions here!
   System.out.println("I am useful!");
}
```

The keyword "void" has a specific meaning for methods in Java. It means they don't return any kind of value. The simply perform a task, and then they are complete.

Return Values with Methods
(returning a variable)

```
class ReturnDemo
{
  public static int daysInWeek()
  {
    int x = 7;
    return x;  // send value back to main
  }

  public static void main(String[] args)
  {
    int days;
    days = daysInWeek(); // call method
    System.out.println(days + "days");
  }
}
```

Return Values with Methods
(using method as a value directly)

```
class ReturnDemo
{
  public static int daysInWeek()
  {
    int x = 7;
    return x;  // send value back to main
  }

  public static void main(String[] args)
  {
    // call method from within println()
    System.out.println(daysInWeek() + "days");
  }
}
```