Making Your Own Methods

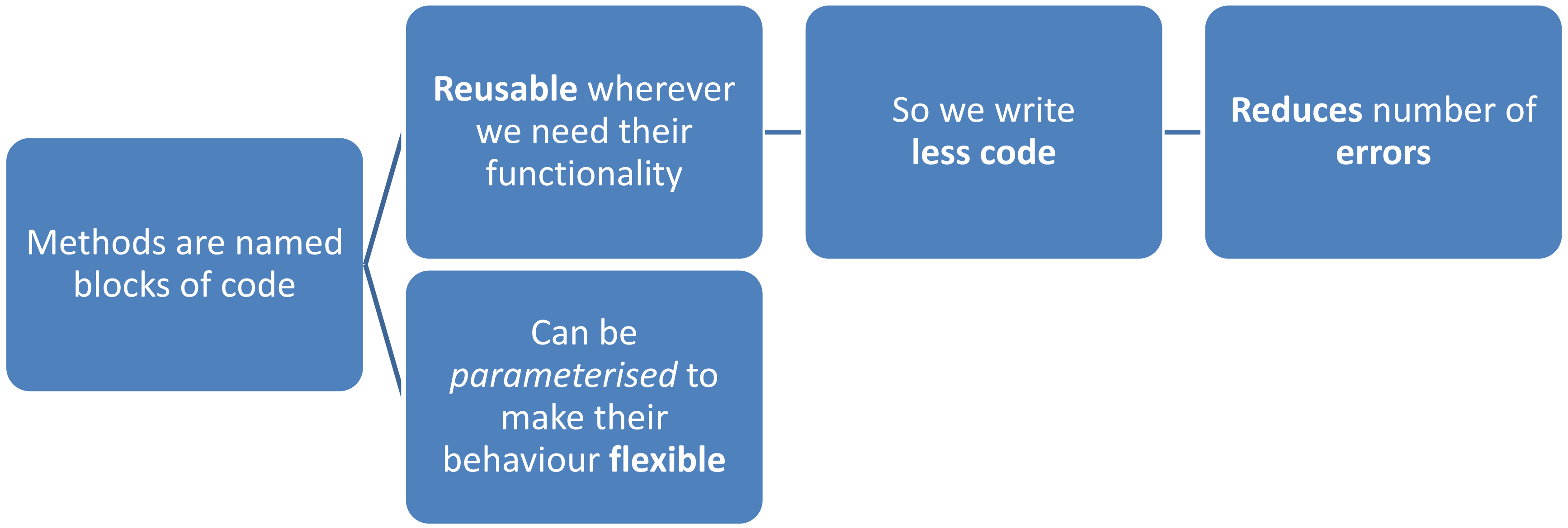
…in single source file programs

class and object
method ●
control structure
statement

07 Methods in Self-contained Programs

Dr James Montgomery, james.montgomery@utas.edu.au

Why do we create methods?


Methods are named blocks of code
Reusable wherever we need their functionality
So we write **less code**
Reduces number of **errors**
Can be *parameterised* to make their behaviour **flexible**

Method declaration

```
/** comments */                                                    Method
public static return type identifier (parameter list) {
    local variable declarations
    code to do the work
    return statement (if return type is not void)
}
```

An (overly complicated) example:

```
/** Returns the area of a rectangle given its width and height. */
public static int area(int width, int height) {
    int area;
    area = width * height;
    return area;
}
```

When writing a method to call from main()

…answer these questions

1 What does is it achieve? This becomes the header comment

3 What name seems suitable?

public static return type identifier (parameter list) **Method Header**

2 What does it return, if anything?

- **`void`** *(nothing) or*
- *primitive type or class name*

4 What data does it need to do its work?

type identifier, type identifier …

5 What work should it actually do (what code to write)?

Worked Example

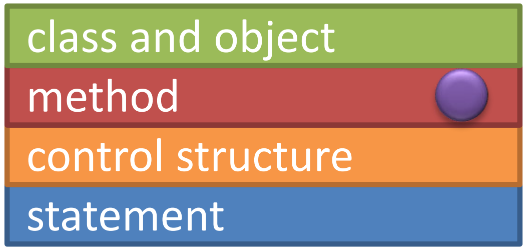
```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int r; //radius of a circle
    double area; //area of that circle

    System.out.println("Hello!");
    System.out.println("Hello!");

    System.out.print("Enter the circle's radius: ");
    r = sc.nextInt();

    area = Math.PI * r * r;
    System.out.println("The area of the circle is " + area);

    System.out.println("Hello!");
    System.out.println("Hello!");
}
```

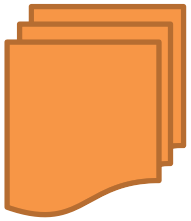
1 We do this action in two places: define a method to make main() simpler

2 A calculation may be reusable: define a method so we can call it as many times as we want

See MethodExample program in Notes: *07 Methods in Self-contained Programs*

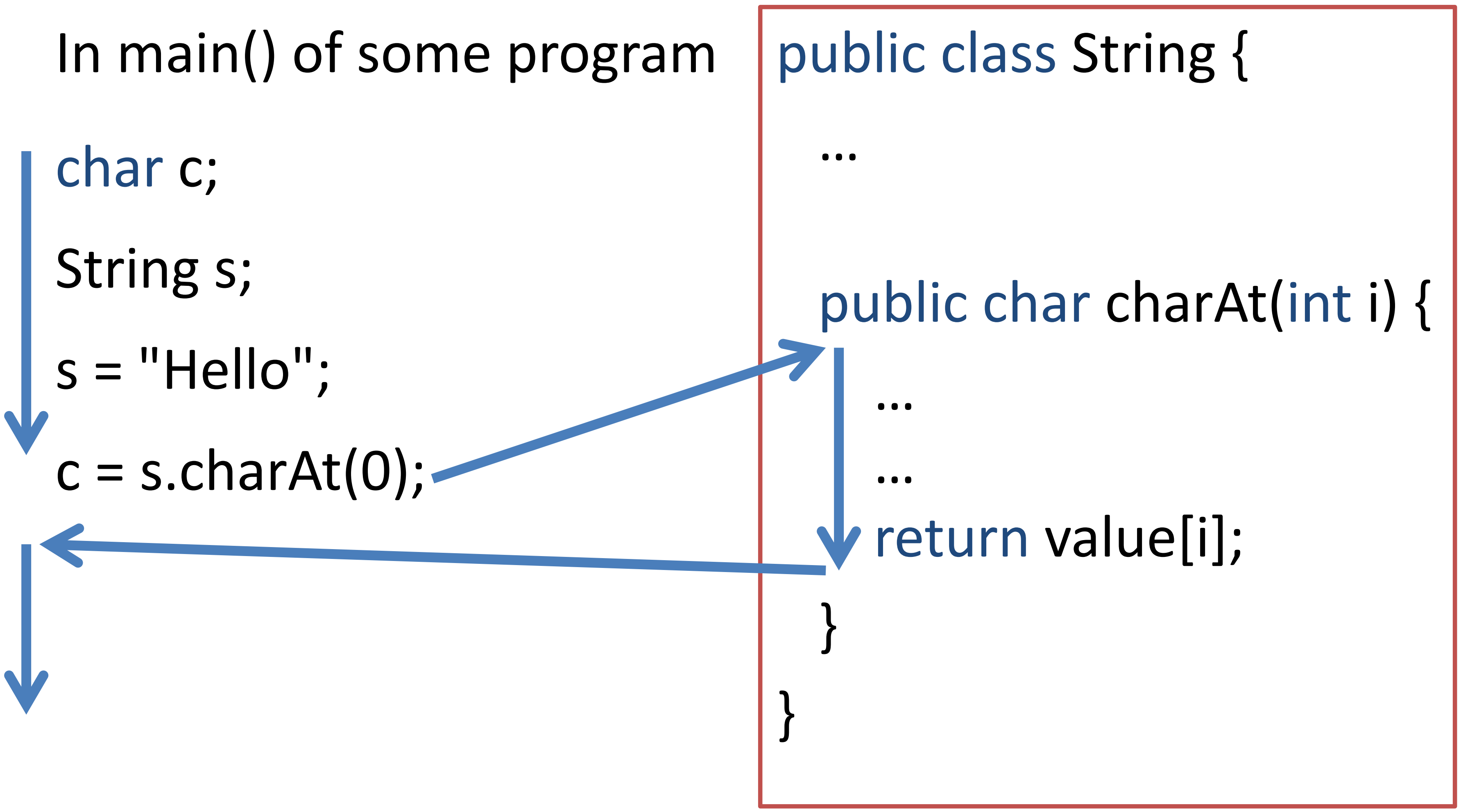
Flow of Control When Calling Methods




07 Methods in Self-contained Programs

Dr James Montgomery, james.montgomery@utas.edu.au

Flow of control with method calls



In main() of some program

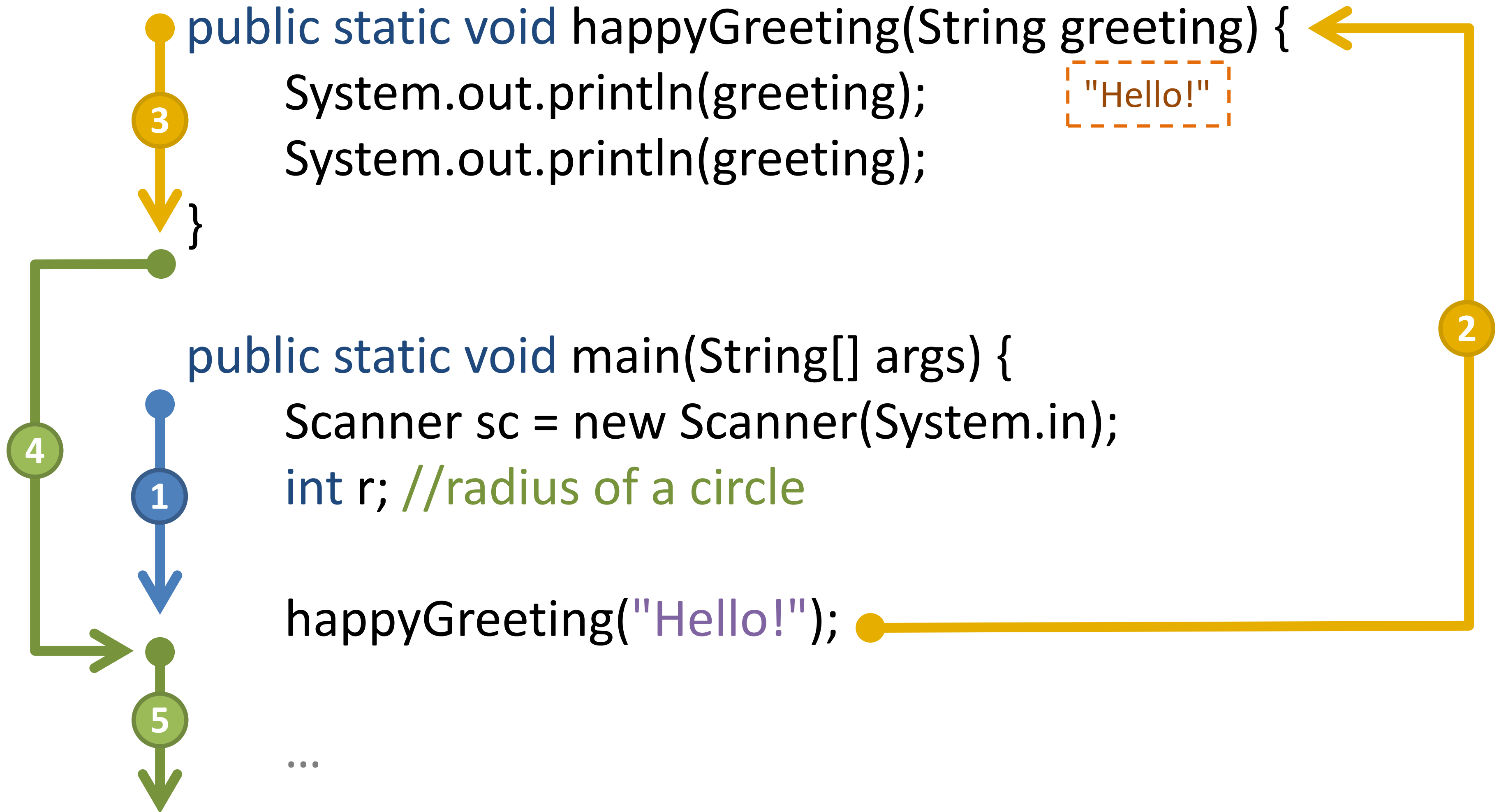
```
char c;
String s;
s = "Hello";
c = s.charAt(0);
```

```
public class String {
   …

   public char charAt(int i) {
      …
      …
      return value[i];
   }
}
```

When a method call is reached, flow of control transfers to that method
The values of any arguments are copied into the parameters of that method

From the MethodProgram example

```
public class MethodProgram {

    /** Prints the greeting twice, over two lines. */
    public static void happyGreeting(String greeting) {
        System.out.println(greeting);
        System.out.println(greeting);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int r; //radius of a circle

        happyGreeting("Hello!");

        ...
```



Flow of data: arguments & parameters

Parameter

- Also called ‘formal parameter’
- Specifies type
- Identifier given by the author

e.g.,

```
public static double
          circleArea(int radius) { …
```

*radius is a **parameter***

Argument

- Also called ‘actual parameter’
- The value passed to method
- May be a variable or literal value
- Variable name **does not** need to match parameter name

e.g.,

```
double a;
a = circleArea(50);
```

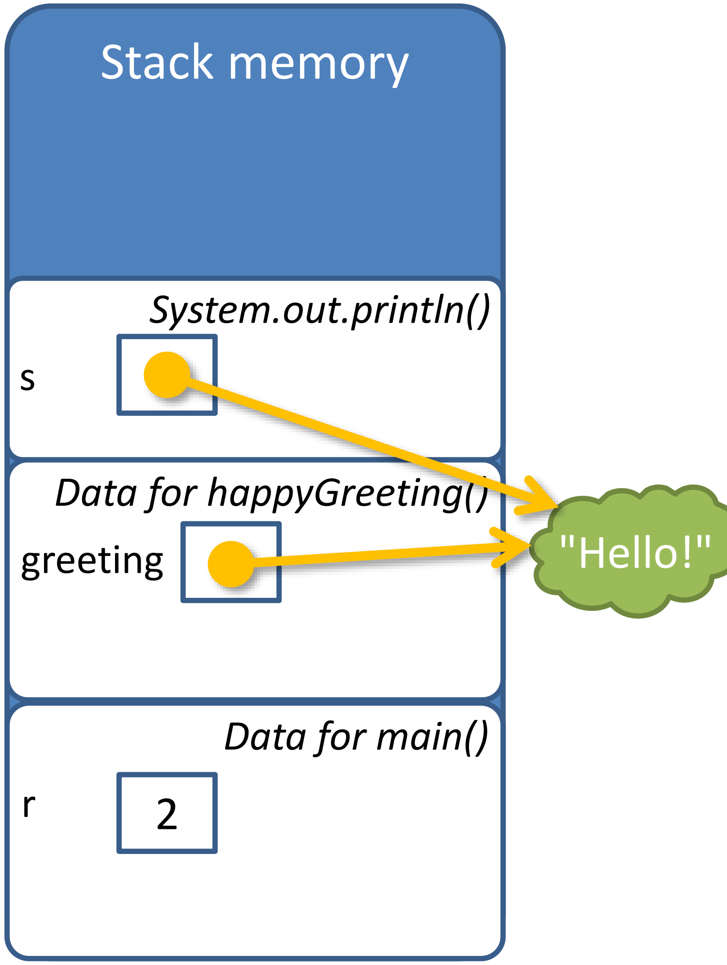
*The int value 50 is an **argument***

Behind the scenes information

It’s OK if this doesn’t make sense yet

Method calls and the stack

```
public class MethodProgramInBrief {

    /** Prints the greeting twice, over two lines. */
    public static void happyGreeting(String greeting) {
        System.out.println(greeting);
        System.out.println(greeting);
    }

    /** Calculates the area of a circle from its radius. */
    public static double circleArea(int radius) {
        return Math.PI * radius * radius;
    }

    public static void main(String[] args) {
        int r = 2; //radius of a circle

        happyGreeting("Hello!");
        System.out.println("area: " + circleArea(r));
    }
}
```

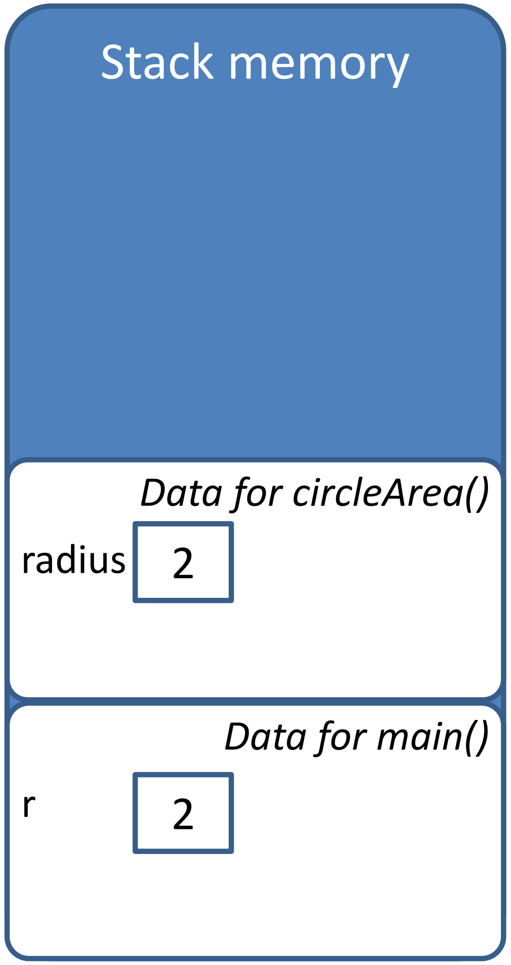


When method call is reached:

- *space allocated for its parameters and local variables*
- *value of argument(s) copied into parameters*

Method calls and the stack

```
public class MethodProgramInBrief {

    /** Prints the greeting twice, over two lines. */
    public static void happyGreeting(String greeting) {
        System.out.println(greeting);
        System.out.println(greeting);
    }

    /** Calculates the area of a circle from its radius. */
    public static double circleArea(int radius) {
        return Math.PI * radius * radius;
    }

    public static void main(String[] args) {
        int r = 2; //radius of a circle

        happyGreeting("Hello!");
        System.out.println("area: " + circleArea(r));
    }
}
```

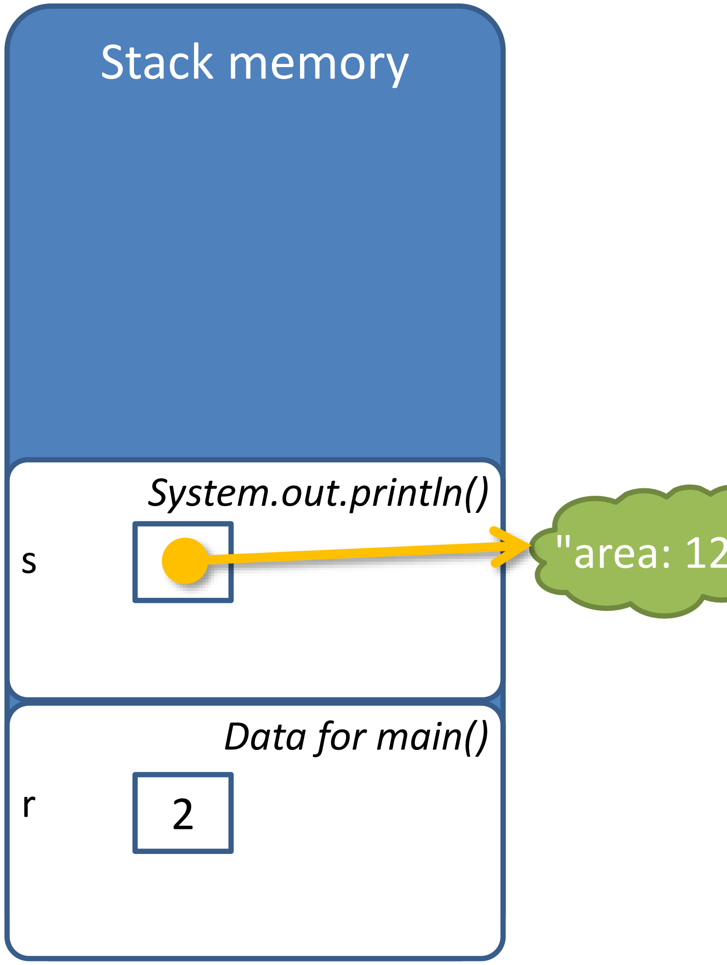


When method call is reached:

- *space allocated for its parameters and local variables*
- *value of argument(s) copied into parameters*

Method calls and the stack

```
public class MethodProgramInBrief {

    /** Prints the greeting twice, over two lines. */
    public static void happyGreeting(String greeting) {
        System.out.println(greeting);
        System.out.println(greeting);
    }

    /** Calculates the area of a circle from its radius. */
    public static double circleArea(int radius) {
        return Math.PI * radius * radius;
    }

    public static void main(String[] args) {
        int r = 2; //radius of a circle

        happyGreeting("Hello!");
        System.out.println("area: " + circleArea(r)); ←
    }
}
```



When method call is reached:

- *space allocated for its parameters and local variables*
- *value of argument(s) copied into parameters*