Garfield AP CS

Parameters

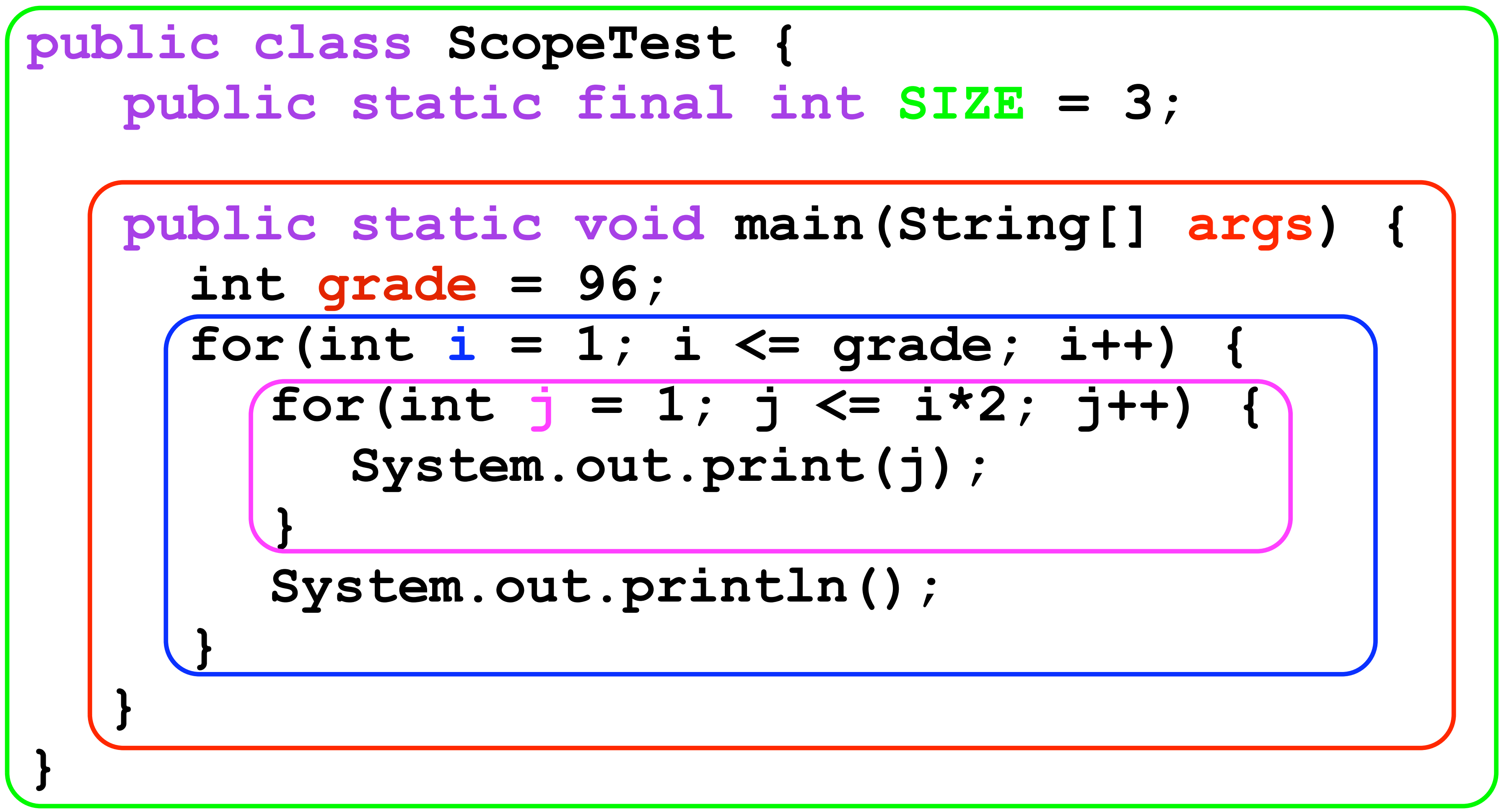
More redundancy

- Looking back at complex figures
  - Code hard to read
  - Redundancy (multiple loops for drawing different numbers of spaces)
- Can variables help?

Scope

- The part of a program in which a declaration is valid
- Loop counter’s scope is limited to loop itself
- Variable declared in main only exists in main

Scope visualized



```
public class ScopeTest {
    public static final int SIZE = 3;

    public static void main(String[] args) {
        int grade = 96;
        for(int i = 1; i <= grade; i++) {
            for(int j = 1; j <= i*2; j++) {
                System.out.print(j);
            }
            System.out.println();
        }
    }
}
```

Local variables

- A local variable is defined in a method so only accessible from that method
- Localizing variables is to declare them in the most local scope possible
- Why localize?
  - Easier to read
  - Less likely to overwrite

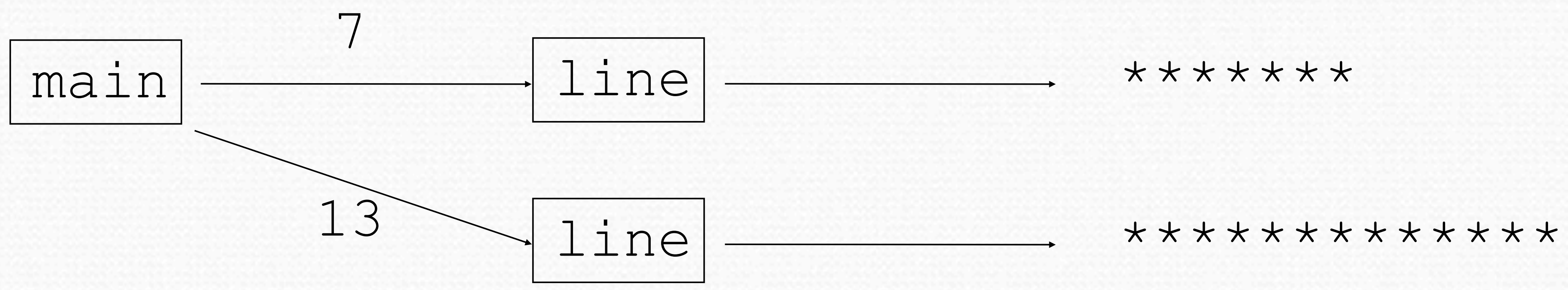
Valid or not?

```
for(int i = 1; i <= 6; i++) {
    for(int i = 1; i <= i + 1; i++) {
        System.out.println(i);
    }
}

for(int i = 1; i <= 6; i++) {
    System.out.println(i);
}
for(int i = 1; i <= 7; i++) {
    System.out.println("Goober!");
}

for(int year = 1; year <= 10; year++) {
    int salary = year * 1000;
}
System.out.println(salary);
```

Parameterization

- **parameter**: A value passed to a method by its caller.

  - Instead of `lineOf7`, `lineOf13`, write `line` to draw any length.
    - When *declaring* the method, we will state that it requires a parameter for the number of stars.
    - When *calling* the method, we will specify how many stars to draw.



Declaring a parameter

Stating that a method requires a parameter in order to run

```
public static void name ( type name ) {
    statement(s);
}
```

- Example:

```
public static void sayPassword(int code) {
    System.out.println("The password is: " + code);
}
```

  - When `sayPassword` is called, the caller must specify the integer code to print.

Passing Parameters

Calling a method and specifying values for its parameters

name (**expression**);

- Example:

```
public static void main(String[] args) {
    sayPassword(42);
    sayPassword(12345);
}
```

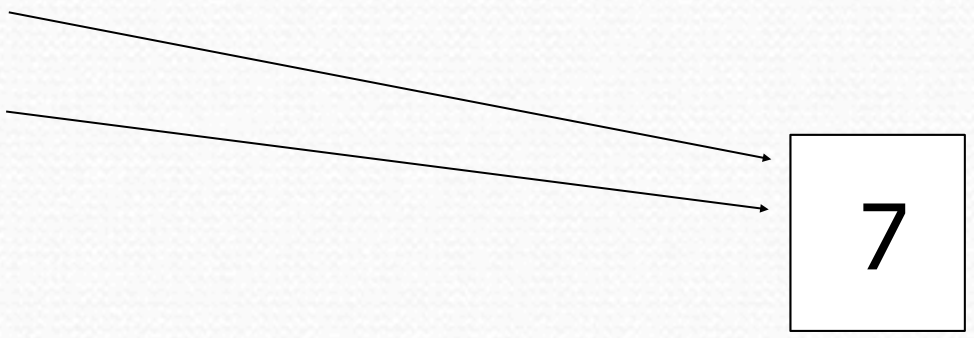
Output:

```
The password is 42
The password is 12345
```

How parameters are passed

- When the method is called:
  - The value is stored into the parameter variable.
  - The method's code executes using that value.

```
public static void main(String[] args) {
    chant(3);
    chant(7);
}

public static void chant(int times) {
    for (int i = 1; i <= times; i++) {
        System.out.println("Just a salad...");
    }
}
```



Common errors

- Not passing a required parameter
- Passing a parameter of the wrong type
- Not using the parameter

Multiple parameters

- The list must be comma-separated
- When calling the method, parameters must be supplied in order

Value semantics

- Primitive types passed in as parameters get values copied
- Modifying the parameter in the method body does not affect the original variable

Parameter mystery

- Great way to make sure you understand parameters

```
public class ParameterMystery {
    public static void main(String[] args) {
        int x = 5;
        int y = 9;
        int z = 2;

        mystery(z, y, x);

        mystery(y, x, z);
    }



    public static void mystery(int x, int z, int y) {
        System.out.println(z + " " + y + " " + x);
    }
}
```

Strings

- **string**: A sequence of text characters.

```
String name = "text";
String name = expression;
```

  - Examples:

```
String name = "Marla Singer";
```

```
int x = 3;
int y = 5;
String point = "(" + x + ", " + y + ")";
```

Strings as parameters

- Modify stars to use strings