

Methods and Parameters

What are methods good for?

- ❖ **Package a unit of code**
 - A well-defined unit of work is named and packaged
 - If well named, aids in higher level design where method name becomes a proxy for the work.
- ❖ **Avoids Repetitive code**
 - Often sections of code are repeated or almost repeated many times
 - It is often possible to define a method to handle that section of code s:
 - Write once
 - Use many times
 - Even if repeated code is not identical, can often make code flexible enough to handle all situations

How do methods communicate?

- ❖ **Method needs to communicate/share info with the rest of the program**
 - Instance variable provide for some of this.
 - Instance variables are “known” throughout the class
 - Parameter provide another way to get info to the method (without “broadcasting” it to the rest of the class.)
 - Return statements provide a way to get info out
- ❖ **Parameters and return each have their limitations**

Parameters

- ❖ **Parameters provide for communications**
 - Always work to get info *into* method
 - For primitive arguments, this is in only
 - Objects passed as parameters may allow info to get *out*
 - *IF*: using mutator methods allows us to change the object
 - *OR*: if object is array, use index specify change to an individual cell
 - Cannot change object like Strings
 - Strings immutable (and have no mutator methods)

Return Statement

- ❖ **Return allows info to be copied out**
 - Invoking statement or expression can use the result
 - Often result to assigned with an =
 - Result may be used in an expression or as an argument to a method

Examples to Illustrate use of Methods

- ❖ **Return allows info to be copied out**
 - Invoking statement or expression can use the result
 - Often result to assigned with an =
 - Result may be used in an expression or as an argument to a method
- ❖ **Remember**
 - “Parameter” in the method header that defines
 - “Argument” when using a method

Examples to Illustrate use of Methods

❖ // from wackadot

```
// from handleCollisions
if (dot.intersects(blueDot)) {
    repositionRandomly(blueDot);
    if (dot.getColor().equals(Color.BLUE))
    {
        dot.setColor(Color.RED);
        score++;
    }
    else
    {
        score--;
    }
    updateString(scoreSprite, "Score: ",
    score);
}
```

```
if (dot.intersects(redDot)){
    repositionRandomly(redDot);
    if (dot.getColor().equals(Color.RED))
    {
        dot.setColor(Color.BLUE);
        score++;
    }
    else
    {
        score--;
    }
    updateString(scoreSprite, "Score: ",
    score);
```

Replace with

Write:

```
private void checkCollision (Sprite badDot, Color old, Color new){  
    if (dot.intersects(badDot)) {  
        repositionRandomly(badDot);  
        if (dot.getColor().equals(old)) {  
            ot.setColor(new);  
            score++;  
        }  
        else {  
            score--;  
        }  
        updateString(scoreSprite, "Score: ", score);  
    }  
}
```

Replace two **if** clauses on previous slide with:

```
checkCollision(blueDot, Color.BLUE, Color.RED);  
checkCollision(redDot, Color.RED, Color.BLUE);
```