Methods
Friday Four Square Today!

Gates, 4:15PM
An Interesting Radio Show

This American Life: “Mr. Daisey and the Apple Factory”

Each point \( k \) is connected to point \( k + 2 \), after wrapping around.

Point \( k \) is at \( \left( x + r \cos \theta, y - r \sin \theta \right) \)

where \( r \) is the radius of the circle, \( x, y \) are the coordinates of the center of the circle, and \( \theta \) is the angle in radians.
Passing Parameters

- A method can accept **parameters** when it is called.
- Syntax:
  ```java
  private void name(parameters) {
      /* ... method body ... */
  }
  ``
- The values of the parameters inside the method are set when the method is called.
- The values of the parameters can vary between calls.
For more on the geometry and properties of stars:

Factorials

• The number **n factorial**, denoted **n!**, is
  \[ 1 \times 2 \times 3 \times \ldots \times (n - 1) \times n \]

• For example:
  • \(3! = 1 \times 2 \times 3 = 6\).
  • \(5! = 1 \times 2 \times 3 \times 4 \times 5 = 120\)
  • \(0! = 1\) (by definition)

• Factorials show up everywhere:
  • Taylor series.
  • Counting ways to shuffle a deck of cards.
  • Determining how quickly computers can sort values.
Returning Values

- A method may produce a value that can be read by its caller.
- To indicate that a method returns a value, specify the type returned in the method declaration:
  
  ```java
  private type name(parameters) {
      /* ... method body ... */
  }
  ```
- A value can be returned with the `return` statement:
  ```java
  return value;
  ```
Subtleties of return

- If a method has non-`void` return type, it must always return a value.

```java
private int thisIsWrong(int x) {
    if (x == 5) {
        return 0;
    }
}
```

What do we return if `x != 5`?
Subtleties of `return`

- If a method has non-`void` return type, it must always return a value.

```java
private int thisIsLegal(int x) {
    if (x == 5) {
        return 0;
    } else {
        return 1;
    }
}
```
Many Happy returns

- A method may have multiple return statements. The method ends as soon as `return` is executed.

```java
private int thisIsLegal(int x) {
    if (x == 5) {
        return 0;
    } else {
        return 1;
    }
}
```
Many Happy \textit{returns} \begin{itemize}
\item A method may have multiple \texttt{return} statements. The method ends as soon as \texttt{return} is executed.
\end{itemize} 
\begin{lstlisting}[language=Java]
private int thisIsLegal(int x) {
    if (x == 5) {
        \texttt{return 0;}
    }
    \texttt{return 1;}
}
\end{lstlisting}

The only way we can get here is if \( x \) is not equal to 5.
Scope

• Each variable has a **scope** where it can be accessed and how long it lives.

```java
for (int i = 0; i < 5; i++) {
    int y = i * 4;
}
i = 3; // Error!
y = 2; // Error!
```
Scope of Method Calls

- A variable declared inside a method is called a **local variable**.
- Local variables can only be accessed inside of the method that declares them.

```java
public void run() {
    int x = 5;
    someOtherMethod();
}

private void someOtherMethod() {
    x = 4; // Error!
}
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

Console Program
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

i 0
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

---

Console Program

---

Slides by Mehran Sahami
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

i 0
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

i 0
```java
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

Console Program
```java
public void run() {
    for (int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

The console program outputs the factorial of each number from 0 to MAX_NUM.
```java
public void run() {
    for (int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

Slides by Mehran Sahami
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

public void run() {
    for (int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

Slides by Mehran Sahami
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

i 0

1

Console Program
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

i  1

Console Program

0! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

i = 1

Console Program

0! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
```java
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

Console Program

0! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

Console Program

0! = 1
public void run() {
    for (int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

0! = 1
public void run() {
    for (int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

Console Program

0! = 1
```java
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}
```

Console Program

```
0! = 1
```
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

n 1 result 1 i 2

Console Program

0! = 1
private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

0! = 1
public void run() {
    for (int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

private int factorial(int n) {
    int result = 1;
    for (int i = 1; i <= n; i++) {
        result *= i;
    }
    return result;
}

0! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

1

Console Program

0! = 1
```java
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

*Console Program*

0! = 1  
1! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

```
0! = 1
1! = 1
```

Console Program

Slides by Mehran Sahami
```java
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

$0! = 1$

$1! = 1$
public void run() {
    for(int i = 0; i < MAX NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

Console Program

0! = 1
1! = 1
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

Console Program

0! = 1
1! = 1
public void run() {
    for (int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
```java
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

Console Program

0! = 1
1! = 1
2! = 2
```
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

i

0! = 1
1! = 1
2! = 2
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

0! = 1
1! = 1
2! = 2
```java
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

```
Console Program

0! = 1
1! = 1
2! = 2
```
```java
public void run() {
    for (int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

Console Program

```
0! = 1
1! = 1
2! = 2
```
```java
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}
```

Slides by Mehran Sahami

Console Program

0! = 1
1! = 1
2! = 2
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

Console Program

0! = 1
1! = 1
2! = 2
3! = 6
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

```
0! = 1
1! = 1
2! = 2
3! = 6
```

Console Program

Slides by Mehran Sahami
public void run() {
    for(int i = 0; i < MAX_NUM; i++) {
        println(i + "! = " + factorial(i));
    }
}

3! = 6

Console Program
0! = 1
1! = 1
2! = 2
3! = 6
Retiring Young
Pass-by-Value

• Java methods pass their parameters by value.
• The method gets a copy of its parameters, not the actual parameters themselves.

```java
private void myMethod(int x) {
    x = 137;
}
```
```
public void run() {
    int x = 42;
    myMethod(x);
    println("The value of x is "+ x);
}
```
This statement prints 42, not 137.
Slowing Things Down
The *pause* Method

- The `pause` method has the signature
  ```java
  public void pause(double milliseconds);
  ```
- *pause* waits the specified number of milliseconds, then returns.
- Examples:
  - `pause(1000);` waits for one second
  - `pause(50);` waits for one twentieth of a second.
Operations on the GObject Class

The following operations apply to all GObject:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object.setColor(color)</code></td>
<td>Sets the color of the object to the specified color constant.</td>
</tr>
<tr>
<td><code>object.setLocation(x, y)</code></td>
<td>Changes the location of the object to the point (x, y).</td>
</tr>
<tr>
<td><code>object.move(dx, dy)</code></td>
<td>Moves the object on the screen by adding dx and dy to its current coordinates.</td>
</tr>
</tbody>
</table>

Standard color names defined in the java.awt package:

- `Color.BLACK`
- `Color.DARK_GRAY`
- `Color.GRAY`
- `Color.LIGHT_GRAY`
- `Color.WHITE`
- `Color.RED`
- `Color.YELLOW`
- `Color.GREEN`
- `Color.CYAN`
- `Color.BLUE`
- `Color.MAGENTA`
- `Color.ORANGE`
- `Color.PINK`
Operations on the **GObject** Class

The following operations apply to all **GObjects**:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>object.setColor(color)</code></td>
<td>Sets the color of the object to the specified color constant.</td>
</tr>
<tr>
<td><code>object.setLocation(x, y)</code></td>
<td>Changes the location of the object to the point ((x, y)).</td>
</tr>
<tr>
<td><code>object.move(dx, dy)</code></td>
<td>Moves the object on the screen by adding (dx) and (dy) to its current coordinates.</td>
</tr>
</tbody>
</table>

Standard color names defined in the `javax.awt` package:

- `Color.BLACK`
- `Color.DARK_GRAY`
- `Color.GRAY`
- `Color.LIGHT_GRAY`
- `Color.WHITE`
- `Color.RED`
- `Color.YELLOW`
- `Color.GREEN`
- `Color.CYAN`
- `Color.BLUE`
- `Color.MAGENTA`
- `Color.ORANGE`
- `Color.PINK`

Graphic courtesy of Eric Roberts
Animation

• By repositioning objects after they have been added to the canvas, we can create animations.

• General pattern for animation:

  ```
  while (not-done-condition) {
    update graphics
    pause(pause-time);
  }
  ```
Physics Simulation
http://physbam.stanford.edu/~fedkiw/animations/motion_smoke.avi
Note that $\Delta y$ increases because the object is accelerating downward.
Let's Code It Up!