CS 112 Introduction to Programming

Animation (sleep and double buffering);

Methods with Return;

Yang (Richard) Yang
Computer Science Department
Yale University
308A Watson, Phone: 432-6400
Email: yry@cs.yale.edu

repeats: Without Method

```java
int N = 5;
for (int line = 1; line <= N; line++) {
    for (int j = 1; j <= (-1 * line + N); j++) {
        System.out.print(".");
    }
    System.out.print(line);
    for (int j = 1; j <= (line - 1); j++) {
        System.out.print(".");
    }
    System.out.println();
}
```

repeats: Using Method

```java
public static void repeats(int n, String p) {
    for (int i = 1; i <= n; i++) {
        System.out.print(p);
    }
}

public static void main(String[] args) {
    int N = 5;
    for (int line = 1; line <= N; line++) {
        repeats(-1 * line + N, ".");
        System.out.print(line);
        repeats(line - 1, ".");
        System.out.println();
    } // end of outer for loop
}
```

Recap: drawCar (Many Magic Numbers)

```java
public static void drawCar(int x0, int y0) {
    StdDraw.setPenColor(Color.BLACK);
    StdDraw.filledRectangle(x0 + 100/2, y0 + 50/2, 100/2, 50/2);
    // draw wheels
    StdDraw.setPenColor(Color.RED);
    StdDraw.filledCircle(x0 + 15 + 10, y0, 10);
    StdDraw.filledCircle(x0 + 100 - 15 - 10, y0, 10);
    // draw window
    StdDraw.setPenColor(Color.Cyan);
    StdDraw.filledRectangle(x0 + 100 - 30 / 2, y0 + 25, 15, 10);
}
```

Recap: drawCar (Use Variable Names)

```java
public static void drawCar(int x0, int y0) {
    // Define the variables to avoid magic numbers
    // A more general version of drawCar may make
    // some into method parameters
    final int CAR_WIDTH = 100, CAR_HEIGHT = 50;
    final int WHEEL_MARGIN = 15, WHEEL_RADIUS = 10;
    final int WINDOW_WIDTH = 30, WINDOW_HEIGHT = 20;
    // Black body
    StdDraw.setPenColor(Color.BLACK);
    StdDraw.filledRectangle(x0 + CAR_WIDTH / 2, y0 + CAR_HEIGHT / 2, CAR_WIDTH / 2, CAR_HEIGHT / 2);
    // Two wheels
    StdDraw.setPenColor(Color.RED);
    StdDraw.filledCircle(x0 + WHEEL_MARGIN + WHEEL_RADIUS, y0, WHEEL_RADIUS);
    StdDraw.filledCircle(x0 + CAR_WIDTH - WHEEL_MARGIN - WHEEL_RADIUS, y0, WHEEL_RADIUS);
    // Window
    StdDraw.setPenColor(Color.Cyan);
    StdDraw.filledRectangle(x0 + CAR_WIDTH - WINDOW_WIDTH / 2, y0 + CAR_HEIGHT / 2, WINDOW_WIDTH / 2, WINDOW_HEIGHT / 2);
}
```
Recap: CarLaunch

// You must have StdAudio.java and race-car.wav in the
// same directory and first compile StdAudio.java.
StdAudio.loop("race-car.wav");

// set up the initial state of the two cars
int h1 = 600, v1x = 30, v1y = 20;
int h2 = 500, v2x = 40, v2y = 30;

// Simulate time from 0 to 10 sec.
for (double t = 0; t < 10; t += 0.1) {
    // Compute car 1's position
double x1 = v1x * t;
double y1 = h1 + v1y * t - 0.5 * 9.81 * t * t;
    // Compute car 2's position
double x2 = v2x * t;
double y2 = h2 + v2y * t - 0.5 * 9.81 * t * t;
    // Used the method defined in Car.java
    // You can also define the method in this file
    Car.drawCar((int)x1, (int)y1);
    Car.drawCar((int)x2, (int)y2);
} // end of for

Checking Time Elapsed

long T0 = System.currentTimeMillis();
...
long diff = System.currentTimeMillis() - T0;

http://docs.oracle.com/javase/7/docs/api/java/lang/System.html#currentTimeMillis()
Exercise: Add a Countdown Scene

- Count down from 10 to 0 and then start the race.

```java
public static void sceneStart(int h1, int h2) {
    for (int t = 10; t >= 0; t--) {
        Car.drawCar(0, h1);
        Car.drawCar(0, h2);
        StdDraw.text(WIDTH/2, HEIGHT/2, ""+t);
        StdDraw.show(1000);
        StdDraw.clear();
    }
}
```

CarLaunch: Remaining Problem

```java
int h1 = 600, v1x = 30, v1y = 20;
int h2 = 500, v2x = 40, v2y = 28;
for (double t = 0; t < 10; t += 0.03) {
    double x1 = v1x * t;
    double y1 = h1 + v1y * t - 9.81 * t * t / 2;
    double x2 = v2x * t;
    double y2 = h2 + v2y * t - 9.81 * t * t / 2;
    Car.drawCar((int)x1, (int)y1);
    Car.drawCar((int)x2, (int)y2);
    StdDraw.show(30); // hold the image for 30 ms
    StdDraw.clear();
}
```

Different Styles of Methods

- "Action oriented methods": External effects, e.g., print, draw, audio
- "Question oriented methods": e.g., what is user’s input of speed? How much is sqrt(n)?
- "Mixed methods": do both

Method with Return: Examples

- Interactive program:
  - The Scanner class provides many methods to return input to your program.

- Math computation:
  - The Math class defines many methods to compute values of common functions

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.abs(value)</td>
<td>absolute value</td>
</tr>
<tr>
<td>Math.ceil(value)</td>
<td>rounds up</td>
</tr>
<tr>
<td>Math.floor(value)</td>
<td>rounds down</td>
</tr>
<tr>
<td>Math.log10(value)</td>
<td>logarithm, base 10</td>
</tr>
<tr>
<td>Math.max(value1, value2)</td>
<td>larger of two values</td>
</tr>
<tr>
<td>Math.min(value1, value2)</td>
<td>smaller of two values</td>
</tr>
<tr>
<td>Math.pow(base, exp)</td>
<td>base to the exp power</td>
</tr>
<tr>
<td>Math.random()</td>
<td>random double between 0 and 1</td>
</tr>
<tr>
<td>Math.sqrt(value)</td>
<td>square root</td>
</tr>
<tr>
<td>Math.sin(value)</td>
<td>sine/cosine/tangent of an angle in radians</td>
</tr>
<tr>
<td>Math.cos(value)</td>
<td></td>
</tr>
<tr>
<td>Math.tan(value)</td>
<td></td>
</tr>
<tr>
<td>Math.toDegrees(value)</td>
<td>convert degrees to radians and back</td>
</tr>
<tr>
<td>Math.PI</td>
<td>Constant Description</td>
</tr>
<tr>
<td>Math.E</td>
<td>Description</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Math Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simply calling math methods produces no visible result.</td>
</tr>
<tr>
<td>Math.pow(3, 4); // no output</td>
</tr>
<tr>
<td>Math methods do useful work by returning values</td>
</tr>
<tr>
<td>To see the result, we must print or store the returned value:</td>
</tr>
<tr>
<td>System.out.println(Math.pow(3, 4)); // 81.0</td>
</tr>
<tr>
<td>double result = Math.pow(3, 4);</td>
</tr>
<tr>
<td>System.out.println(result); // 81.0</td>
</tr>
</tbody>
</table>
Return vs Parameter

- Return is the opposite of a parameter:
  - Parameters send information **in** from the caller to the method.
  - Return value sends information **out** from a method to its caller.

```
Math.abs(-42)  // 42
Math.abs(-3)   // 3
Math.round(2.71) // 3
```

Why return and not print?

- It might seem more useful for the Math methods to print their results rather than returning them. Why don’t they?

```
// We can compute several things before printing:
double pow1 = Math.pow(3, 4);
double pow2 = Math.pow(10, 4);
System.out.println("Powers are " + pow1 + " and " + pow2);

// We can combine the results of many computations:
double k = 13 * Math.pow(3, 4) + 5 - Math.sqrt(17.8);
```

Math Questions

- Evaluate the following expressions:
  - Math.abs(-1.23)
  - Math.toRadians(90)
  - Math.abs(Math.min(-3, -5))
  - Math.random()

- Consider an int variable named `age`.
  - What expression would replace negative ages with 0?
    - Math.max(age, 0)
  - What expression would cap the maximum age to 25?
    - Math.min(age, 25)

Defining a Method Returning a Value

```
public static double fToC(double degreesF) {
    double degreesC = (degreesF - 32) * 5.0 / 9.0;
    return degreesC;
}
```

More return

- The return type of a method indicates the type of value that the method sends back to the calling location.
  - a method that does not return a value has a void return type

- The return statement specifies the value that will be returned:
  - its expression must conform to the return type
  - if you define a non-void method, you must return a valid type expression
  - there can be multiple return statements to return (finish running) at multiple points
A Common Error

Many people incorrectly think that a return statement sends a variable’s name back to the calling method.

```java
public static void main(String[] args) {
    fToC(60);
    System.out.println("60F = " + result);
}

public static double fToC(double degreesF) {
    double result = 5.0 / 9.0 * (degreesF - 32);
    return result;
}
```

Fixing the Common Error

Instead, returning sends the variable’s value back. The returned value must be stored into a variable or used in an expression to be useful to the caller.

```java
public static void main(String[] args) {
    double c = fToC(65);
    System.out.println("65F = " + c + "C");
    System.out.println("Again, 65F = " + fToC(65) + "C");
}
```

Exercise: Revise CarLaunch

Revise CarLaunchV2 to use a method with return

```java
public static double pos(double initPos, double speed, double a, double t) {
    return initPos + speed * t + a * t * t / 2;
}
```

Exercise Solution

```java
public static double pos(double initPos, double speed, double a, double t) {
    return initPos + speed * t + a * t * t / 2;
}
```

Method “Puzzle”:

```java
int h1 = 600, v1x = 30, v1y = 20;
int h2 = 500, v2x = 40, v2y = 28;
for (double t = 0; t < 10; t += 0.1) {
    double y1 = pos(h1, v1y, -9.81, t);
    // Parameter type mismatch. Can we use the method?
}
```

Method “Puzzle” II:

```java
System.out.println(Math.round(10.3));
```

Two definitions of same method name?

// Math.round() has two definitions
// definition 1
static long round(double a)
// definition 2
static int round(float a)
Method Definition/Invocation Rules

- **Definition rule:**
  - You can define multiple methods with the same name in a class. This is called method *overloading*.
  - To distinguish different overloaded methods, these methods must have different *signatures*.
  - The signature is the sequential list of the type of each parameter.

- **Invocation rule:**
  - Java compiler picks the best matched method allowed by implicit conversion.

Overloaded Methods

```
Version 1: signature: int
double tryMe (int x) {
    return x + .375;
}
```

```
Version 2: signature: int_double
double tryMe (int x, double y) {
    return x * y;
}
```

```
Version 3: signature: double_int
double tryMe (double x, int y) {
    return x * y;
}
```

```
Version 4: signature: double_double
double tryMe (double x, double y) {
    return x * y;
}
```

Invocation

```
result = tryMe (25, 4.32)
```

Overloading Picks the Best Match allowed by Implicit Conversion

```
double tryMe ( int x )
{
    return x + 5;
}
```

```
double tryMe ( double x )
{
    return x * .375;
}
```

```
double tryMe ( double x, int y)
{
    return x + y;
}
```

```
Which tryMe will be called?
```

Overload Matching only Signature

```
int x = (int)Math.round(10.3);
```

```
int x = Math.round(10.3);
```

```
ERROR: Type mismatch.
I know 10 will fit as an int: how do I change from long to int?
```

```
int x = (int) Math.round(10.3);
```

```
int x = Math.round(10.3);
```

```
// Math.round() has two definitions
// definition 1
static long round(double a)
// definition 2
static int round(float a)
```

Question: Interactive CarLaunch

- How to extend the CarLaunch program to get input from user on initial parameters:
  - h1, v1x, v1y
  - h2, v2x, v2y
  - sound file to play