CS 112 Introduction to Programming

Methods with Return;
Summary of Method Definition and Invocation

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson, Phone: 432-6400
Email: yry@cs.yale.edu
Outline

- Admin and recap
- Method details
  - Method w/ return
  - Summary of method definition and invocation rules
    - Overloaded methods
    - Formal arguments are local variables
    - Primitive types use value semantics
Admin

- PS3 status and questions?
Recap: Animation

```c
// FRAME_T = 60;
int h1 = 541, v1x = 50, v1y = 30;
int h2 = 381, v2x = 30, v2y = 10;

for (double t = 0; t < 10; t += FRAME_T/1000.0) {
    double x1 = v1x * t;
    double x2 = v2x * t;

    double y1 = h1 + v1y * t - 9.81 * t * t / 2;
    double y2 = h2 + v2y * t - 9.81 * t * t / 2;

    Car.drawCar(x1, y1, CAR1_SIZE);
    Car.drawCar(x2, y2, CAR2_SIZE);
}

StdDraw.show(FRAME_T); // hold image for 30 ms
StdDraw.clear(); // now clear up
```
Animation Internal: Use Both Screen (Video Memory) and (Double) Buffer

In animation mode, drawing is in the (double) buffer.

Next show(t) statement copies to screen.

When finished drawing, the double buffer is copied to video memory.
for (double t = 0; t < 10; t += FRAME_TIME/1000.0) {
    double x1 = v1x * t;
    double x2 = v2x * t;

    double y1 = H1 + V1y * t - 9.81 * t * t / 2;
    double y2 = H2 + V2y * t - 9.81 * t * t / 2;

    Car.drawCar( x1, y1, CAR1_SIZE );
    Car.drawCar( x2, y2, CAR2_SIZE );

    StdDraw.show(FRAME_TIME); // hold image for 60 ms
    StdDraw.clear(); // now clear up
}
Outline

- Admin and recap
- Method details
  - Method w/ return
**Different Styles of Methods**

“Action-oriented methods”: External effects (print, drawing, audio), e.g.,
drawCar(0,0,10), drawCar(10, 10, 20), …

“Answer-oriented methods”: e.g., how much is \sqrt{2}, \sqrt{10}?

“Mixed methods”: do both
Method with Return

- Math methods are good examples of “answer-oriented” methods: they do useful work by returning values
# Example: Math Methods

<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.round(value)</td>
<td>nearest whole number</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.toRadians(value)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.E</td>
<td>2.7182818...</td>
</tr>
<tr>
<td>Math.PI</td>
<td>3.1415926...</td>
</tr>
</tbody>
</table>
**Math Methods**

- Simply calling math methods produces no visible result.
  - Math.pow(3, 4); // no output

- To see the result, we must print or store the returned value:
  - System.out.println( Math.pow(3, 4) ); // 81.0
  - double result = Math.pow(3, 4);
  - System.out.println(result); // 81.0
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?

**Answer:** Returning is more flexible than printing.

- We can compute several things before printing:
  ```java
  double pow1 = Math.pow(3, 4);
  double pow2 = Math.pow(10, 6);
  System.out.println("Powers are "+pow1+" and "+pow2);
  ```

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

Evaluate the following expressions:

- `Math.abs(-1.23)`
- `Math.toRadians(180)`
- `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)`

http://download.oracle.com/javase/6/docs/api/java/lang/Math.html
Defining a Method Returning a Value

```java
public static <type> <name>(<parameters>) {
    <statements>
    ...
    return <expression>;
}
```
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.

```
main

Math.abs(-42)

Math.round(2.71)

-42
42
2.71
3
```
Exercise: Fahrenheit -> Celsius

- Write method f2c that converts from Fahrenheit to Celsius, both in double
// Converts degrees Fahrenheit to Celsius.
public static double f2c(double degreesF) {
    double degreesC = (degreesF - 32) * 5.0 / 9.0;
    return degreesC;
}

You can shorten the example by returning an expression:

public static double fToC(double degreesF) {
    return (degreesF - 32) * 5.0 / 9.0;
}
Exercise: Revise CarLaunch

- Revise CarLaunch to use a method with return
Exercise Solution

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

// ERROR: result not defined
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");
}

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

- Design a loan program to compute the monthly amortization table of a fixed-rate loan

http://en.wikipedia.org/wiki/Mortgage_calculator


Loan.java
Rules of Fixed-Rate Loan

- Assume N periods (e.g., 120 months)

- For each period, borrower pays interest on the remaining owed (principal) at the fixed rate

- At the end of N's period, the remaining principal goes to 0
**Fixed-Rate Loan Calculation Alg.**

- Alg. focuses on owed (principal)  
  - $p$: principal
  - $m$: monthly payment
  - $r$: monthly interest rate

Owed at initiation: $p$

Owed after 1 month: $(1 + r)p - m$

Owed after 2 month: $(1 + r)[(1 + r)p - m] - m$

  $= (1 + r)^2 p - [1 + (1 + r)]m$

Owed after 3 month: $(1 + r)[(1 + r)^2 p - [1 + (1 + r)]m] - m$

  $= (1 + r)^3 p - [1 + (1 + r) + (1 + r)^2]m$
Owed after N month: \((1+r)^N p - \left[1 + (1+r) + \ldots + (1+r)^{N-1}\right] m\)

apply \(1 + x + \ldots + x^{N-1} = \frac{x^N - 1}{x - 1}\)

Owed after N month: \((1+r)^N p - \frac{(1+r)^N - 1}{r} m\)

Payoff loan after N month => \((1+r)^N p - \frac{(1+r)^N - 1}{r} m = 0\)

\[
\frac{(1+r)^N - 1}{r} m = (1+r)^N p
\]

\[
m = \frac{r(1+r)^N}{(1+r)^N - 1} p
\]
Outline

- Admin and recap
- Method details
  - Method w/ return
  - Summary of method definition and invocation rules
Summary: Method Definition

- Why define methods?
  - Denote structure, eliminate redundancy
  - A method with parameters solves an entire class of similar problems
  - A method with return gives back an answer on a question
Many methods in Math has multiple definitions: https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

System.out.print( Math.round(10.3) );

// Math.round() has two definitions

// definition 1
static long round(double a)

// definition 2
static int round(float a)
Method Definition and 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 method with the best match in signature, allowed by implicit conversion.
Overloaded Methods

Version 1: signature: int

```java
double tryMe (int x)
{
    return x + .375;
}
```

Invocation

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

Version 2: signature: int_double

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

Version 3: signature: double_int

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

Version 4: signature: double_double

```java
double tryMe (double x, double y)
{
    return x * y;
}
```
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?

tryMe( 1 );

tryMe( 1.0 );

tryMe( 1.0, 2 );

tryMe( 1, 2 );

tryMe( 1.0, 2.0 );
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?

// Math.round() has two definitions

// definition 1
static long round(double a)

// definition 2
static int round(float a)
Outline

- Admin and recap
- Method details
  - Method w/ return
  - Summary of method definition and invocation rules
    - Overloaded methods
    - Formal arguments are local variables
Method Invocation and Parameter Passing

- Corresponding actual *argument* in the invocation is assigned to the corresponding *formal argument*

```java
public static void printNumber(int number, int count)
{
    for (int i = 1; i <= count; i++) {
        System.out.print(number);
    }
    System.out.println();
}
```

```java
int line = 3;
printNumber(line-1,5);
```

---

```java
public static void printNumber(int number, int count)
{
    // equiv: number = 2; count = 5;
    for (int i = 1; i <= count; i++) {
        System.out.print(number);
    }
    System.out.println();
}
```
Method Invocation and Parameter Passing

- In Java, a formal argument is a local variable of a method.
- The formal argument and the actual argument are different variables, with different memory locations, even if they have the same name.
public class ParameterMystery {
    public static void main(String[] args) {
        int x = 9;
        int y = 2;
        int z = 5;

        mystery(z, y, x);
        mystery(y, x, z+y);
    }
}

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

- Admin and recap
- Method details
  - Method w/ return
  - Summary of method definition and invocation rules
    - Overloaded methods
    - Formal arguments are local variables
    - Primitive types use value semantics
Method Invocation and Parameter Passing

- When a primitive variable is passed as the actual argument to a formal argument, the value is copied
  - Value copying implies value semantic
  - Implication: modifying the parameter inside the method will not affect the variable passed in.
Value Semantics

```java
int a = 100;
double x = 45.12;
```

A value variable stores a value of the type of the variable.
Value Variables

int a = 100;
double x = 45.12;
int aa;
An assignment of one value variable to another value variable copies the value.
Value-Variable Assignment

int a = 100;
double x = 45.12;
int aa;
aa = a;
a = 200;

Change the value of one value variable will not change the other.
public static void strange(int x) {
    x = x + 1;
    System.out.println("1. x = " + x);
}

public static void main(String[] args) {
    int x = 23;
    strange(x);
    System.out.println("2. x = " + x);
    ...
}

Output:
1. x = 24
2. x = 23