CS 112 Introduction to Programming

Java Graphics: Examples of Parameterized Drawing and Loops

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

- Admin and recap
- Parameterized graphics methods and loops
Admin

- PS3 walk-through and coding style review session
  - today or tomorrow (Thursday)?
Recap: Java Graphics

- Java provides a large number of methods for graphics
- A graphical method may need to use a large number of parameters
- We will use StdDraw, which provides a wrapper class with a large number of drawing methods
  - How to use StdDraw: see comments of http://zoo.cs.yale.edu/classes/cs112/cs112-2017-spring/examples/StdDraw/SimpleStdDrawX.java
Outline

- Admin and recap
- Java graphics
  - Coordinate system and basic shape
  - Drawing w/ color
What if we want to draw the X in red?

- StdDraw has two methods to set pen color
  - `StdDraw.setPenColor(R, G, B);`
  - `StdDraw.setPenColor(Color);`
    - predefined class constants defined in the `Color` class
Exercise: SimpleStdDrawXColor

- Modify SimpleStdDrawX to use `setPenColor(R, G, B)` to draw the lines red
Class Constants

- **class constant**: A static class variable with a fixed value
  - value can be set only at declaration; cannot be reassigned

- **Syntax**:
  ```java
  public static final type name = value; // in class scope
  ```
  - name is usually in ALL_UPPER_CASE

- **Examples**:
  ```java
  public static final int DAYS_IN_WEEK = 7;
  public static final double INTEREST_RATE = 3.5;
  public static final int SSN = 658234569;
  ```
Java predefines many class constants in the Color class:

```
Color.CONSTANT_NAME
```

where CONSTANT_NAME is one of:

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

http://download.oracle.com/javase/8/docs/api/java/awt/Color.html
The Color class is part of Java standard class library.

A class library is a collection of classes that one can use when developing programs.

- Libraries are not part of the Java language per se, but using them is essential to achieve productive programming.
- A big advantage of Java is that it provides a quite large standard class library.
Library and Packages

- The classes in a library are organized into **packages**
  - think of packages as folders, which help you to get organized
- Some of the packages in the standard class library are:

<table>
<thead>
<tr>
<th>Package</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.lang</td>
<td>General support, e.g., Math, String, System</td>
</tr>
<tr>
<td>java.applet</td>
<td>Creating applets for the web</td>
</tr>
<tr>
<td>java.awt</td>
<td>Graphics and graphical user interfaces</td>
</tr>
<tr>
<td>javax.swing</td>
<td>Additional graphics capabilities and components</td>
</tr>
<tr>
<td>java.net</td>
<td>Network communication</td>
</tr>
<tr>
<td>java.util</td>
<td>Utilities</td>
</tr>
<tr>
<td>java.text</td>
<td>Text processing</td>
</tr>
</tbody>
</table>

- Color belongs to a package named `java.awt`

http://download.oracle.com/javase/7/docs/api/
The **import** Declaration

- When you want to use a class from a package, you could use its **fully qualified class name**, e.g.,
  ```java
  java.awt.Color color;
  ```
  Or you can **import** the class, then just use the class name
  ```java
  // put this at the very top of your program
  import java.awt.Color;
  ```

- To import all classes in a particular package, you can use the ** wildcard character**
  ```java
  // put this at the very top of your program
  import java.awt.*;
  ```
Example: Using Colors

- **Pass a Color to** `StdDraw's setPenColor` method
  - Subsequent shapes will be drawn in the new color.

```java
import java.awt.Color;
StdDraw.setPenColor(Color.BLUE);
StdDraw.line(20, 0, 10, 30);

StdDraw.setPenColor(Color.RED);
StdDraw.line(20, 0, 10, 30);
```
Exercise: SimpleStdDrawX

- Modify SimpleStdDrawX to draw the lines red using Color.RED
Outline

- Admin and recap
- Java graphics
  - Coordinate system and basic shape
  - Drawing w/ color
  - Parameterized drawing examples
Exercise: Parameterized Drawing

- Write method drawCar(x0, y0, size):
  - size = 100
  - Center: (x0 + 0.5 size, y0 + 0.25 size)
  - Size: 0.5 size, 0.25 size
  - Center: (x0 + (1.00 - 0.30/2) size, y0 + 0.50/2 size)
  - Size: 0.15 size, 0.10 size
  - Center: (x0 + (1.00 - 0.15 - 0.10) size, y0)
  - Size: 0.10 size

Car.java
Exercise: Book Cover
(Color and Loop)

- White 500x600 drawing panel
- Three components at
  - (20, 415), (165, 330), (220, 85)
    with sizes 150, 120, and 240
  - Each component
    - Yale blue background
    - white "CS112" text left @ 1/2 horizontal, 4/5 vertical
    - 10 brown (red=192, green=128, blue=64) "bricks"
      - 2 pixel between two adjacent bricks
Backup: Method Details
Exercise: Parameterized Drawing

Target.java
Outline

Method details

- Method parameters: value semantics and variables in a method
- Method signatures: overloaded methods
Example: What is the Output?

```java
class StrangeExample {
    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
```
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.

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;

a \quad 100
x \quad 45.12
aa
Value-Variable Assignment

An assignment of one value variable to another value variable copies the value.
Value-Variable Assignment

```c
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.
Example: `main()` start

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

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

```java
public static void strange(int x) {
    x = x + 1;
    System.out.println("1. x = " + x);
}
```

compiler declares formal argument `x` and copies value from the actual argument
Example: Local update

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

```java
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);
}

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);
}
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);
    }

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

A "Parameter Mystery" problem
Java Variable Storage

- A variable of the primitive type (e.g., int, char, double) **stores value**
  - thus primitive type variables are also called **value variables**

- A variable of a non-primitive type **stores reference** to the actual content (one-level of indirection)
  - thus non-primitive-type variables are also called **reference variables**

- Value variables ≠ reference variables
Outline

- Method details
  - Method parameters: value semantics and variables in a method
  - Method signatures: overloaded methods
Method “Puzzle”:

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)

Two definitions of same method name?
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

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

Invocation

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

Version 2: signature: int_double

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

Version 3: signature: double_int

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

Version 4: signature: double_double

```c
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 );
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?

// Math.round() has two definitions

// definition 1
static long round(double a)

// definition 2
static int round(float a)
Constructing Objects

- An object is created from a **class**

- Constructing (creating) an object:
  \[
  \text{Type objectName} = \text{new Type(parameters)};
  \]

- Calling an object's method:
  \[
  \text{objectName.methodName(parameters)};
  \]
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
}
```