CS 112 Introduction to Programming

Summary of Methods; User Input using Scanner

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

Part 1: speed and angle interpretation
Walkthroughs
Monday and Tuesday evenings
Debugging session:
Please contribute buggy code to us
Midterm 1 date: Mar. 3

Recap: StdDraw.show(T)

Display
Buffer

Recap: General Method Definition

public static type name(parameters) {
  statements;
  ...
  return expression;
}

Summary: Method Definition

- Why define methods?
  - Denote structure, eliminate redundancy
  - A method with parameters solves an entire class of similar problems
- Can you define in the same class multiple methods with the same name?
  - Yes. This is called method overloading, as long as the overloaded methods must have different signatures, where the signature of a method is the sequential list of the type of each parameter

Summary: Method Invocation (I)

- How does the compiler pick the method to use for overloaded methods?
  - The compiler picks the method according to signature match.
Summary: Method Invocation (II)

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

```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();
}
```

```java
int line = 3;
printNumber(line-1, 5);
// equiv: number = 2; count = 5;
```

Formal Arguments are Local Variables

- 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

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

<table>
<thead>
<tr>
<th>a</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>45.12</td>
</tr>
</tbody>
</table>

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

Value Variables

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

<table>
<thead>
<tr>
<th>a</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>45.12</td>
</tr>
<tr>
<td>aa</td>
<td></td>
</tr>
</tbody>
</table>

Value-Variable Assignment

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

<table>
<thead>
<tr>
<th>a</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>45.12</td>
</tr>
<tr>
<td>aa</td>
<td>100</td>
</tr>
</tbody>
</table>

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;
```

<table>
<thead>
<tr>
<th>a</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>45.12</td>
</tr>
<tr>
<td>aa</td>
<td>100</td>
</tr>
</tbody>
</table>

Change the value of one value variable will not change the other.
Exercise: What is the output?

```java
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);
}
```

Example: main() start

```java
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);
}
```

Example: Local update

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

Example: Method return

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

Example: Method return

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

```java
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 z, int y) {
        System.out.println(z + " and " + (y - x));
    }
}
```

What is the output?

Summary: 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.

Outline

- Admin and recap
- Text input using methods from the Scanner class

Interactive Programs

- Interactive programs can be easier to use and have more interesting behavior.
- Interactive programs can be tricky: users are unpredictable and may misbehave.
- Java text input is based on the Scanner class.

Some Scanner Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextInt()</td>
<td>Returns an int from source</td>
</tr>
<tr>
<td>nextDouble()</td>
<td>Returns a double from source</td>
</tr>
<tr>
<td>next()</td>
<td>Returns a one-word string from source</td>
</tr>
<tr>
<td>nextLine()</td>
<td>Returns a one-line string from source</td>
</tr>
</tbody>
</table>
Problem of using **Scanner**

- It is common that the same program reads input simultaneously from multiple sources:
  - `System.in` (the opposite of `System.out`)
  - Files, strings, web sites, databases, ...

Design Option I

**Method**

<table>
<thead>
<tr>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Scanner.nextInt(&lt;src&gt;)</code></td>
</tr>
<tr>
<td><code>Scanner.nextDouble(&lt;src&gt;)</code></td>
</tr>
<tr>
<td><code>Scanner.next(&lt;src&gt;)</code></td>
</tr>
<tr>
<td><code>Scanner.nextLine(&lt;src&gt;)</code></td>
</tr>
</tbody>
</table>

Design Option II: **Objects (briefly)**

- **object**: An entity that contains both data and behavior.
  - **data**: variables inside the object
  - **behavior**: methods offered by the object
  - You interact with the methods; most data are hidden in the object.

Constructing Objects

- An object is created from a class
- Constructing (creating) an object by calling the constructor method:
  ```java
type objectName = new Type(parameters);
```
- Calling an object’s method:
  ```java
objectName.methodName(parameters);
```

Packages

- The classes in Java are organized into **packages**
  - think of packages as folders, which help you to get organized
- Some of the packages in Java 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>

- Scanner belongs to the **java.util** package

The **import** Declaration

- When you want to use a class from a non-default / `java.lang` package, you could use its **fully qualified class name**, e.g.,
  ```java
  java.util.Scanner console;
  ```
- Or you can **import** the class, then just use the class name
  ```java
  // put this at the very top of your program
  import java.util.Scanner;
  Scanner console;
  ```
- 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.util.*;
  ```
Using Scanner

```java
import java.util.Scanner;
...
Scanner console = new Scanner(System.in);

// Typically print a prompt
System.out.print("How old are you? ");
int age = console.nextInt();
System.out.println("You typed "+ age);
```

Using System.in is to interact using the Terminal:

![Terminal with prompt and output]

Scanner Example

```java
import java.util.*; // so that I can use Scanner
public class UserScannerInput {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Which year will you graduate? ");
        int year = console.nextInt();
        int rYears = year - 2014;
        System.out.println(rYears + " years remaining at Yale!");
    }
}
```

Console (user input underlined):
Which year will you graduate?
0 years remaining at Yale!

UserScannerInput.java

Scanner Example 2

```java
The Scanner can read multiple values from one line.
```n
```java
import java.util.*; // so that I can use Scanner
public class ScannerMultiply {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("Please type two numbers: ");
        int num1 = console.nextInt();
        int num2 = console.nextInt();
        int product = num1 * num2;
        System.out.println("The product is "+ product);
    }
}
```

Output (user input underlined):
Please type two numbers: 8 6
The product is 48

ScannerMultiply.java

Scanning Details

- The OS will not send input to Scanner constructed using System.in until user hits enter
- `nextInt()`, `nextDouble()`, `next()` are token based scanning methods
  - skip whitespace (spaces, tabs, new lines) until find first non-white space, collect input into a token until a whitespace, send token to the method to interpret: the following white space remains
  - How many tokens appear on the following line of input?
    - 23  John Smith   42.0  "Hello world"  $2.50  "  19"
- `nextLine()` collects any input character into a string until the first new line and discards the new line

ScannerTokenDiff.java

Practice: Scanner Fun

- Please try out ScannerFun.java
Input from File

- There are two approaches
  - Create a scanner with src as a file (more later)
  - Redirect a file as standard input
    (command line)
    ```
    %java PlotUSA < USA.txt
    ```

PlotUSA.java  USA.txt

Design Issue

- What value to return when a token is not
  the type the scanner expects

```java
System.out.print("What is your age? ");
int age = console.nextInt();
```

Output:
What is your age? Timmy

Token and Exception

- When a token is not the type that the scanner
  expects, since no reasonable (non-ambiguous)
  return value, Scanner throws an exception
  (panic)

```java
System.out.print("What is your age? ");
int age = console.nextInt();
```

Output:
What is your age? Timmy

java.util.InputMismatchException
at java.util.Scanner.nextInt (Unknown Source)
at java.util.Scanner.nextInt (Unknown Source) ...

Why Not a “Smarter” nextInt()?

- For example, continue to scan the input to
  find the integer?

- Design principle: design of basic methods
  should KISS (Keep It Simple and Stupid)
The if Statement

Executes a block of statements only if a test is true

```java
if (test) {
    statement;
    ...
    statement;
}
```

- Example:
  ```java
  if (grade >= 90.0 && grade <= 100) {
    System.out.println("It is an A.");
  }
  ```

The if/else Statement

- An else clause can be added to an if statement to make it an if-else statement:
  ```java
  if (test) {
    statement1;
  } else {
    statement2;
  }
  ```
- If the condition is true, statement1 is executed; if the condition is false, statement2 is executed
- One or the other will be executed, but not both

Example:
```java
if (gpa >= 2.0 && gpa <= 3.8) {
    System.out.println("Welcome to Middle Univ.!");
} else {
    System.out.println("Application denied.");
}
```

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

Mapping Loop Variable

\[\begin{align*}
p & : \text{principal} \\
m & : \text{monthly payment} \\
r & : \text{monthly interest rate}
\end{align*}\]

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:
\[
(1+r)^N p - \frac{(1+r)^N - 1}{r} m = 0
\]

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

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