CS 112 Introduction to Programming

Text Input/Output; Object (briefly); Scanner

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

- Admin and recap
- Method w/ return
- Summary of method definition and invocation rules
  - overloaded methods
  - formal arguments are local variables
  - primitive types use value semantics
- Text I/O
  - Input: basic Scanner input
  - Output: basic printf and String.format
Admin

- PS4 walk-through?
Recap: Defining a Method Returning a Value

```java
public static type name(parameters) {
    statements;
    ...
    return expression;
}
```
Recap: Method Definition, Invocation, Formal Arguments (Parameters)

Definition rule
- You can define multiple methods with the same name in a class. This is called **method overloading**.
- These methods must have different **signatures**.

Invocation
- Compiler picks the **best match**, allowed by implicit conversion
- Values of actual arguments are copied to formal arguments, which are **local variables**
- All primitive variables are value variables, with **value semantics**
Recap: Value-Variable Assignment and Value Semantics

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

Explanation: main() start
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);
}
Explanation: 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);
}
```

```
int x = 23;
strange(x);
System.out.println("2. 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);
}

Example: Method return
any program you might want to write

- objects
- methods and classes
- graphics, sound, and image I/O
- arrays
- conditionals and loops
- math
- text I/O
- primitive data types
- assignment statements
Motivation: CarLaunch with User Input

- Extend the CarLaunch program to get input from user on initial parameters:
  - $h_1$, $v_{1x}$, $v_{1y}$
  - $h_2$, $v_{2x}$, $v_{2y}$
  - sound file to play

- Benefit:
  - Change program parameters without need to change the program and recompilation

- Issue: Interactive programs can be tricky: users are unpredictable and may misbehave.

- Java user input is based on the Scanner class
## Some `Scanner` Methods

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

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

Problem: A lot of redundancy in program,

```java
h1 = Scanner.nextDouble("System.in");
v1x = Scanner.nextDouble("System.in");
v1y = Scanner.nextDouble("System.in");
h2 = Scanner.nextDouble("System.in");
v2x = Scanner.nextDouble("System.in");
v2y = Scanner.nextDouble("System.in");
soundFile = Scanner.next("System.in");
```
Design Option II: Objects (briefly)

- **object**: An entity that contains both data (state) and behavior.
  - **Data (state)**
    - variables inside the object to remember state
  - **behavior**
    - methods offered by the object
  - You interact with the methods; most data (state) are hidden in the object (think of object methods as methods with memory for now).
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);
  ```
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);
```

No need to specify that it is from standard input (keyboard)

Create an object, which remembers that it is for standard input
Scanner using System.in as Input

- Using System.in is to interact using the Terminal:
import java.util.Scanner;  // so that I can use Scanner

public class ScannerInputExample {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        System.out.print("Which year will you graduate? ");
        int gYear = console.nextInt();

        int rYear = gYear - 2017;
        System.out.println(rYear + " more years at Yale!");
    }
}

Console (user input underlined):
Which year will you graduate?
2 more years at Yale!
2018
The Scanner can read multiple values from one line.

```java
import java.util.Scanner;  // 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
The OS will not send input to Scanner constructed using System.in until user hits enter (reason?)

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
Exercise: `nextInt`, `nextLine`

```java
int i1 = console.nextInt();
String s1 = console.nextLine();
```

Result: `i1 = 2018   s1 = ""`

```java
2018←
```

Result: `i1 = 2018   s1 = ""`

```java
☐2018☐☐☐a☐←
```

Result: `i1 = 2018   s1 = "☐☐☐a☐"`
Exercise: `nextInt, next`

```java
int i1 = console.nextInt();
String s1 = console.next();
```

result: `i1 = 2018`  `s1 = "a"`

First read `ScannerTokenDiff.java` to guess behaviors, then try.
Exercise: nextLine or next?

System.out.print("A number: ");
int i1 = console.nextInt();
System.out.print("File name: ");
String fileName = console._____( );
(Offline) Practice: Scanner Fun

- Please try out ScannerFun.java
There are two approaches

- Create a scanner with src as a file (more shortly)

- Redirect: a concept in computer science, which allows the operating system to send a file’s content as if it is typed in by keyboard:
  (Use command line: Terminal -> to working directory)

```bash
%java Plot < USA.txt
```
Exercise: Plot a Geo Data File

- File format:
  - xmin ymin xmax ymax npoints
  - x, y
  - ...

Plot.java  USA.txt
Input from File

PlotUSA.java  USA.txt
Exercise

- What if you do not want to see the loading process (e.g., see all display at once, not one point at a time)?
Design Issue

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

```java
System.out.print("Which year will you graduate? ");
Int year = console.nextInt();

Output:
Which year will you graduate? 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("Which year will you graduate? ");
int year = console.nextInt();
```

Output:
Which year will you graduate? **Timmy**

```java
java.util.InputMismatchException
at java.util.Scanner.nextInt(Unknown Source)
at java.util.Scanner.nextInt(Unknown Source)
...
Issue: How to avoid crash when user may give wrong type of input?

System.out.print("Which year will you graduate? ");
int year = console.nextInt();

Output:
Which year will you graduate? Timmy
Approach 1: Test before Proceed

System.out.print("Which year will you graduate? ");
int year = console.nextInt();

Output:
Which year will you graduate? Timmy

Robust design

1. Add a test method to check whether the input has the expected type
2. if the test failed, report error
The if statement

Executes a block of statements only if a test is true

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

Example:

double gpa = console.nextDouble();
if (gpa >= 3.9)
    System.out.println("Welcome to Yale!");
The *if/else* Statement

- An *else clause* can be added to an *if statement* to make it an *if-else statement*:

  ```
  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.
The **if/else** Statement

- Example:

```java
if (gpa >= 3.8) {
    System.out.println("Welcome to Yale!");
} else {
    System.out.println("We recommend Harvard.");
}
```