CS 112 Introduction to Programming

Text Input/Output; Exceptions; Boolean Conditions

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

- Admin and recap
- Text I/O
  - Input: basic Scanner input
  - Output: basic printf and String.format
- Program flow of control
  - Boolean expressions
Admin

- PS4 walk-through tonight (Monday) at 6:30 pm at DL 220

- Community building: even if you do not need help from the TFs, it helps to work during the office hours at DL 120 so that students can work together.

- Exam 1 date?
Recap: **Scanner**

- **Basic 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>

- **Design methodology:** Java uses an object to remember the state (e.g., source) of a scanner

```java
Scanner console = new Scanner(System.in);
console.nextInt();
```
Recap: Scanning Details

- `nextLine()` collects any input character into a string until the first new line and discards the new line.

- `nextInt()`, `nextDouble()`, `next()` are token based scanning methods:
  - Ignore leading white space characters.
  - Collect non-white space chars into a token.
  - Interpret the token as an int/double for `nextInt/nextDouble`.

- Design issue: What if the token is not in the correct format?
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)

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

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

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

- **Exception**: a programming language mechanism to represent a *runtime error*, e.g.,
  - dividing an integer by 0
  - trying to read the wrong type of value from a `Scanner`
  - trying to read a file that does not exist

- Java has defined a set of exceptions, each with a name, e.g., `ArithmeticException`, `InputMismatchException`, `FileNotFoundException`
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)

- Higher level programs handle the case in their specific settings
Two basic approaches

- Test before proceed
- Proceed and clean up (try/catch)
**Robust Input Approach 1: Test Before Proceed**

- **Design pattern**

```java
if ( <ExceptionConditionFalse> ) {
    proceed;
} else {
    System.out.println(“Error message.”);
}
```

The return type of `hasNextInt()` is `boolean`, indicating a logical condition.

```java
System.out.print(“Which year will you graduate? ”);
if ( console.hasNextInt() ) {
    year = console.nextInt();
} else {
    System.out.println(“Wrong type; please give int.”);
}
```
Robust Input Approach 2: try and catch

- **try/catch**: if an exception happens, program execution jumps to the `catch` statement, skipping the rest in the `try` block.

```java
try {
    potentially dangerous statements
}
catch (ExceptionName e) {
    handle exception, such as print an error message
}
// end of catch
```
Robust Input Approach 2: Example

```java
import java.util.Scanner; // for Scanner

public class ScannerInputExampleTry {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        try {
            System.out.print("Which year will you graduate? ");
            int year = console.nextInt();
            System.out.println("Your give " + year);
        } catch (InputMismatchException e) {
            // print an error message
            System.out.println("You give a wrong input type.");
        } // end of catch

    } // end of main
}
```
import java.io.File;       // for File
import java.util.Scanner;  // for Scanner

public class PlotFile {
    public static void main(String[] args) {
        File f = new File("USA.txt");
        Scanner input = new Scanner( f );
        ...
    }
}

- Compilation fails with the following error:
  PlotFile.java:9: unreported exception java.io.FileNotFoundException;
  must be caught or declared to be thrown
    Scanner input = new Scanner( f );
What Happened: the **throws** Clause

- **throws clause**: Keyword on a method's header to state that it may generate an exception (and will not handle it) and those using it **must handle** it (called a **checked** exception; `nextInt` does not declare it).

- **Syntax**:
  
  ```java
  public static <type> <name>(...) throws <type> {
  ```

- **Example**:

  ```java
  http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html#Scanner(java.io.File)
  ```
import java.io.File; // for File
import java.util.Scanner; // for Scanner

class PlotFile {
    public static void main(String[] args) {
        try {
            File f = new File("USA.txt");
            Scanner input = new Scanner(f);

            ...  
        } catch (FileNotFoundException e) { 
            // print an error message
            System.out.println("File not found exception");
        }  // end of catch

    } // end of main
}
Outline

- Admin and recap
- Text I/O
  - Input: Scanner input
    - Scanner using object to remember state
    - Scanner input with exceptions (run time errors)
  - Output: basic printf and String.format
A Tiny Bit History of Java Text Formatting

- Before Java 1.5, Java provides formatting classes such as NumberFormat and DecimalFormat classes as part of the java.text package.

- But many programmers like the more flexible method signature of printf() starting from the C programming language.

- Starting from Java 1.5, printf/formatr is added and typically preferred by many programmers.
Discussion

- Text output formatting as of now
  - String concatenation without ability to specify per variable format ....
System.out.printf("format string", parameters);

- Output with *placeholders* to insert parameters, e.g.,
  - %d integer
  - %f real number
  - %s string
    - these placeholders are used instead of + concatenation

- Example:
  int x = 3;
  int y = -17;
  System.out.printf("x is %d and y is %d!\n", x, y);

    // x is 3 and y is -17!

  *printf does not drop to the next line unless you write \n
https://docs.oracle.com/javase/8/docs/api/java/util/Formatter.html
In the most general case, Java allows flexible (variable) method signature, e.g.,

```java
public static <type> name(<type1> param1, <type2>... param2)
```

Number and type of parameters determined by the first parameter. We will not learn how to define such methods, but will use some: `printf()` and `format()`
printf Width

- %d \quad integer, W characters wide, right-aligned
- %d \quad integer, W characters wide, left-aligned
- %f \quad real number, W characters wide, right-aligned
- ...

```java
for (int i = 1; i <= 3; i++) {
    for (int j = 1; j <= 10; j++) {
        System.out.printf("%4d", (i * j));
    }
    System.out.println();  // to end the line
}
```

Output:

```
  1   2   3   4   5   6   7   8   9  10
  2   4   6   8  10  12  14  16  18  20
  3   6   9  12  15  18  21  24  27  30
```
printf **Precision**

- `%Df` real number, rounded to *D* digits after decimal
- `%W.Df` real number, *W* chars wide, *D* digits after decimal
- `%W.Df` real number, *W* wide (left-align), *D* after decimal

```java
double gpa = 3.253764;
System.out.printf("your GPA is \%.1f\n", gpa);
System.out.printf("more precisely: %8.3f\n", gpa);
```

**Output:**

your GPA is 3.3

more precisely: 3.254
Many more formatting control options supported by `printf`, e.g., using the comma (,) to display numbers with thousands separator

```
System.out.printf("%d\n", 58625);
System.out.printf("%,.2f\n", 12345678.9);
```

Output:

```
58,625
12,345,678.90
```
System.out.printf and String.format

- String.format has the same formatting capability as printf, except that printf outputs and String.format returns:

  System.out.printf("\%,.2f\n", 12345678.9);
  String s = String.format("\%,.2f\n", 12345678.9);
  System.out.print( s );

  Output:
  12,345,678.90
  12,345,678.90
Exercise: F2C

- Revise F2C to print 2 decimal digits.
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
Program Flow of Control

- Java has three types of program flow of control:
  - decision statements, or conditional statements: decide whether or not to execute a particular statement
  - repetition statements, or loop statements: perform a statement over and over repetitively
  - exceptions: to handle run-time errors (atypical)

- The foundation of conditional/loop program flow of control is the logical condition, which should be a boolean expression
Basic Boolean Condition: Relational Comparison

- A basic Boolean expression is to compare two values using a *relational operator*:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Meaning</th>
<th>Example</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>equals</td>
<td>$1 + 1 == 2$</td>
<td>true</td>
</tr>
<tr>
<td>!=</td>
<td>does not equal</td>
<td>$3.2 != 2.5$</td>
<td>true</td>
</tr>
<tr>
<td>&lt;</td>
<td>less than</td>
<td>$10 &lt; 5$</td>
<td>false</td>
</tr>
<tr>
<td>&gt;</td>
<td>greater than</td>
<td>$10 &gt; 5$</td>
<td>true</td>
</tr>
<tr>
<td>&lt;=</td>
<td>less than or equal to</td>
<td>$126 &lt;= 100$</td>
<td>false</td>
</tr>
<tr>
<td>&gt;=</td>
<td>greater than or equal to</td>
<td>$5.0 &gt;= 5.0$</td>
<td>true</td>
</tr>
</tbody>
</table>

- Note the difference between the equality operator (==) and the assignment operator (=)
Example: Basic Boolean Expression

```java
public class Flip {
    public static void main(String[] args) {
        if (Math.random() < 0.5) System.out.println("Heads");
        else System.out.println("Tails");
    }
}
```

Flip.java

```
% java Flip
Heads
% java Flip
Heads
% java Flip
Tails
% java Flip
Heads
```
Example: Chaos Game

- Play on equilateral triangle, with vertices R (node 0), G (node 1), B (node 2)
  - Start at R
  - Repeat N times
    - Pick a random vertex
    - Move halfway between current point and vertex
    - Draw a point in color of chosen vertex
Example: Chaos Game