

Department of Computer Science

Interactive Programming

Dr. Chia-Ling Tsai

Outline

- `System.in`
- Using a scanner
- Sentinels
- Patterns for sentinels
- Scanner lookahead
- Retrying input
- Fancier input method -- `JOptionPane`

System.in

- Java's standard input object is `System.in`, corresponding to the standard output object, `System.out`
- However, `System.in` does not provide useful methods for input, corresponding to the output methods of `System.out`
 - Why not? Bad design
 - You can read a byte at a time, but can't get recognition of integers, decimals, etc.

Constructing A Scanner

- To do input with the convenience of Java's output, you must *construct a scanner object*
 - *Import* the `Scanner` class from the Java utility library
 - Declare a variable to name your scanner
 - Use the `new` keyword to construct a scanner object to initialize that variable
- Once you have a scanner object, you can ask it to get you an integer, a decimal, a string, etc., from the user of the program

Using A Scanner

- A scanner allows you to get user input from the keyboard
 - Actually, the source of the input can be the keyboard or a *file* – *more later*
- You can ask for an integer using the `nextInt` method or a decimal using `nextDouble`
- To read an entire line into a string, use `nextLine`
- Many of the programs we have seen can be improved by asking for input, instead of setting values of variables
 - Instead of changing the program when you want different numbers, just run it again, and input new numbers

Prompting for Input

- The user cannot be expected to read our program (or our minds), so the program must communicate what we expect him or her to do
 - To do this, we use *prompts*
- Input is harder than output, because input must be read and checked, and then stored in the proper way
 - Java already knows what the output will be

Example

```
import java.util.Scanner; ← Import from utility library

public class UseScanner
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Enter an integer, then a decimal");
        int i = keyboard.nextInt();
        double x = keyboard.nextDouble(); } Input statements

        System.out.println("You entered " + i + " and " + x);
    }
}
```

Construct a scanner called keyboard, based on System.in

Prompt

```
// A program that demonstrates interactive I/O

import java.util.Scanner;

public class InteractiveDemo
{
    public static void main(String[] args)
    {
        // Construct a scanner for the keyboard
        Scanner keyboard = new Scanner(System.in);

        // Set value for this year
        int thisYear = 2008;

        // Get information from user
        System.out.print("Hi! What's your name? ");
        String name = keyboard.nextLine();
        System.out.print("How old are you? ");
        int age = keyboard.nextInt();
        System.out.print("How many years ahead do you want to look? ");
        int yearsAhead = keyboard.nextInt();

        // Give user feedback
        System.out.println(name + ", you are now " + age + " years old");
        int futureYear = thisYear + yearsAhead;
        int futureAge = age + yearsAhead;
        System.out.println("On your birthday in " + futureYear
            + ", you will be " + futureAge + " years old");
    }
}
```

Output

(User input is shown in blue)



```
C:\WINDOWS\system32\cmd.exe
Hi! What's your name? Jessica
How old are you? 8
How many years ahead do you want to look? 12
Jessica, you are now 8 years old
On your birthday in 2020, you will be 20 years old
Press any key to continue . . . _
```




In-Class Exercise

- Let's make our GCD program for user friendly.

Sentinels

- One important use of loops is to allow repeated entry of data
- Often, we don't know how much data to expect
- We really ought not ask the user to count the data
- We can ask the user to indicate the end of data in some special way
- A special "end-of-data" value is called a *sentinel*
- Sentinels may be dealt with in several ways

A Pattern For Sentinels

- We have to continue while the incoming value is not the sentinel
- Problem: we can't *phrase* the condition before we have an incoming value
- Sentinel Pattern 1
 - Get a value  *Outside the loop*
 - Loop: while value gotten isn't sentinel
 - Work on value 
 - Get a value 

Sample Program

```
import java.util.Scanner;
public class AddNumbers1
{
    public static void main(String[] args)
    {
        final int SENTINEL = 0;
        Scanner keyboard = new Scanner(System.in);

        System.out.println("Enter integers to be added. End with " + SENTINEL);

        int number = keyboard.nextInt();
        int sum = 0;
        while (number != SENTINEL)
        {
            sum = sum + number;
            number = keyboard.nextInt();
        }
        System.out.println("The sum is " + sum);
    }
}
```

Sentinel Pattern 1

Accumulator Pattern

Another Sentinel Pattern

- Instead of reversing the logic of the loop, declare a **boolean** variable
boolean working = **true**;
- The condition becomes whether or not this variable is true
while (working)
 {
 ...
 }
- No need to reverse logic, but must now reset the variable based on input

Another Sentinel Pattern

■ Sentinel Pattern 2

- Declare a `boolean` to be `true`
- Loop: while variable is true ("working")
 - Get value
 - If value isn't sentinel, work on value, otherwise reset `boolean` variable

■ You can also take the negative approach

- Declare a `boolean` to be `false`
- Loop: while variable is false ("not done")
 - Get value
 - If value isn't sentinel, work on value, otherwise reset `boolean` variable

Samples

```
int sum = 0;
boolean working = true;
while (working)
{
    int number = keyboard.nextInt();
    if (number != SENTINEL)
    {
        sum = sum + number;
    }
    else
    {
        working = false;
    }
}
```

```
int sum = 0;
boolean done = false;
while (!done)
{
    int number = keyboard.nextInt();
    if (number != SENTINEL)
    {
        sum = sum + number;
    }
    else
    {
        done = true;
    }
}
```

Scanner Lookahead

- The `Scanner` class has *lookahead* methods that return `boolean` values
 - `hasNextInt()` – true if the user has entered an `int` value, otherwise false
 - `hasNextDouble()` – true if the user has entered a `double` value, otherwise false
- If we tell the user to enter something that isn't an integer, such as a letter, we can simplify the loop
 - *Note:* You can't ask the user to press enter or the spacebar without complicating things again

Sample Program

```
import java.util.Scanner;
public class AddNumbers3
{
    public static void main(String[] args)
    {
        Scanner keyboard = new Scanner(System.in);

        System.out.println("Enter integers to be added.  End by entering any letter");

        int sum = 0;
        while (keyboard.hasNextInt())
        {
            int number = keyboard.nextInt();
            sum = sum + number;
        }
        System.out.println("The sum is " + sum);
    }
}
```

Retrying Input

- When user input is required and the user enters an inappropriate value, we should let him or her *retry*
- To do this, we use a loop that will execute *only if the user makes an error*
- Pattern (as in Sentinel Pattern 1)
 - Get value *before* loop
 - Execute loop only while value is *not acceptable*
 - Get subsequent values to check *at bottom* of loop
 - When loop finishes (or does not execute) value from user is correct

Sample Program

- A loop that will run only if input is bad, and will continue as long as input remains bad

```

Scanner keyboard = new Scanner(System.in);
System.out.print("Please enter an integer between 1 and 10: ");
// Get the number
int entry = keyboard.nextInt(); ← First entry obtained before loop
// As long as number is not in range, retry
while (entry < 1 || entry > 10)
{
    System.out.println("Sorry. " + entry + " is not in range.");
    System.out.print("Please enter an integer between 1 and 10: ");
    entry = keyboard.nextInt(); ← Subsequent entries obtained
                                at bottom of loop body
}
System.out.println("Thank you.");

```

Repeats as long as entry is no good
If OK the first time, the loop never runs

Our Old Example Revisited

```
// rewrite of the previous version with a scanner
// object for input and sentinel guard for correct input
import java.util.Scanner;
public class PopulationDoubler2 {
    public static void main(String[] args) {
        // Get data from user
        Scanner keyboard = new Scanner(System.in);
        System.out.print("initial population? ");
        int initialPopulation = keyboard.nextInt();
        // Get the rate of growth and make sure it is valid
        System.out.print("Annual rate of growth of the population? ");
        double rate = keyboard.nextDouble();
        while (rate<0) {
            System.out.println("At a rate of "+ rate+" the population is shrinking!");
            System.out.print("Annual rate of growth of the population? ");
            rate = keyboard.nextDouble();
        }
        // Calculate number of years needed to double size
        int years = 0;
        int population = initialPopulation;
        while (population < 2*initialPopulation) {
            years++;
            double increase = rate * population;
            population = (int) (population + increase);
        }
        System.out.println("The population will double in "+years+" years.");
    }
}
```

Fancier Input Method

- It would be nice to have a dialogbox for the input.
- This can be easily achieved if we use *JOptionPane*, instead of the *Scanner*
- An exception is thrown if the input is invalid
 - We better catch it and handle it!
 - try {
 - // code to execute under normal condition
 - }
 - catch (exception-classname variable-name) {
 - // code to take care of the unexpected

AddNumbers Revisited

```

import javax.swing.*;

// Adding number with nice dialog windows
public class AddNumbers2 {
    public static void main(String[] args) {
        final int SENTINEL = 0;
        int number = -1;
        int sum = 0;
        System.out.println("Enter 0 to exit.");
        do {
            String numLine = JOptionPane.showInputDialog("Enter the integer to be added: ");
            try {
                number = Integer.parseInt(numLine);
                sum += number;
                System.out.println("Accumulative sum = "+sum);
            }
            catch (NumberFormatException e) {
                JOptionPane.showMessageDialog(null, "Not a number!");
            }
        } while (number != SENTINEL);

        System.out.println("Final sum = "+sum);
    }
}

```

Something might go wrong here

Exception handler

In-class exercise

- Let's rewrite the program using the scanner object.

Summary

- Reading input from the command window using a scanner
- Reading input with a dialog window
- Using sentinels for repeated input values
- Error handling