ICS 111
Introduction to Computer Science I

Nikki Manuel
Leeward Community College

Output, Math Functions, Strings, and Handling Errors

Week 4
Fall 2019

Output

Formatting Output

Conversions & Typecasting

Using `printf`

Conversions & Typecasting

Java Numeric Data Types

Java numeric data types:

- `byte` (-128 to +127)
- `short` (-32,768 to +32,767)
- `int` (--2 billion to +2 billion, approximately)
- `long` (-9E18 to +9E18, approximately)

- `float` (-3.4E38 to +3.4E38)
- `double` (-1.7E308 to +1.7E308)

Data Conversion

Keep in mind:

- Java is strongly-typed
- Variables are declared with a specific type

When mixing numbers within an arithmetic expression, the results may be unexpected.

Type casting

Type casting is an operation that converts a value of one data type into a value of a different data type.

Widening/Narrowing Conversion

When you **widen a type**, you go from a type with a smaller range to a type with a larger range.

byte → short → int → long → float → double

When you **narrow a type**, you go from a type with a larger range to a type with a smaller range.

double → float → long → int → short → byte

Type Casting Syntax

Java will automatically *widen* a type,
but you must explicitly narrow a type.

The syntax for casting a type is to specify the target type in parentheses, followed by the variable's name or the value to be cast:

```
(targetType)variableName
```

Type Casting Syntax Examples

```
System.out.print((int)1.7);
```

`Output: 1` // when double → int, the fraction part is removed

```
System.out.print((double)1/2);
```

`Output: .5` // 1 is cast to 1.0, then 1.0 is divided by 2

Remember:

```
System.out.print(1/2);
```

`Output: 0` // because 1 & 2 are integers, so result is also int

Type Casting: Important Note!

Casting does not change the variable being cast.

For example, **x** is not changed after casting:

```
double x = 4.5;
int y = (int)x;                    // y becomes 4, but x is still 4.5
```

Using `printf`

printf

You can use the **`System.out.printf`** method to display output that you specifically format

`printf` Example: Why do we use it?

The following code computes interest, given the amount and annual interest rate:

```
double amount = 12618.98;
double interestRate = 0.0013;
double interest = amount * interestRate;
System.out.println("Interest is $" + interest);
```

```
Interest is $16.404674
```

`printf` Example: How we use it!

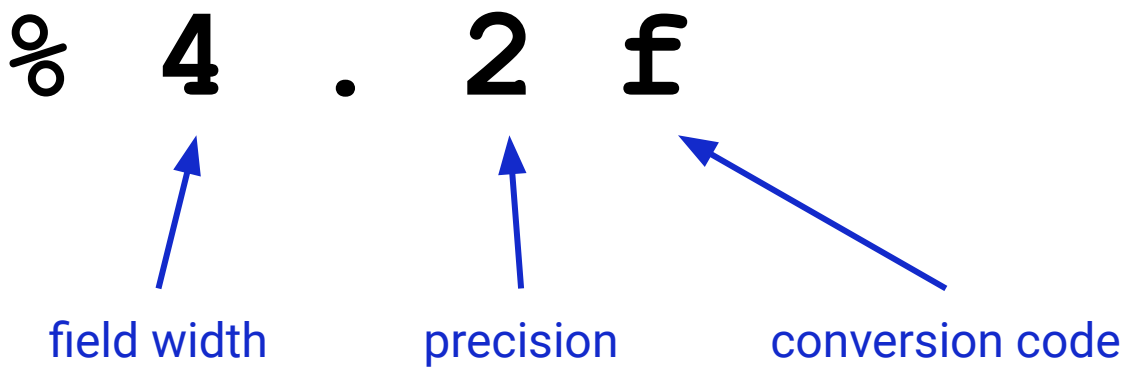
The following code computes interest, given the amount and annual interest rate:

```
double amount = 12618.98;
double interestRate = 0.0013;
double interest = amount * interestRate;
System.out.printf("Interest is $%4.2f", interest);
```

```
Interest is $16.40
```

`printf` Example: How we use it!

```
System.out.printf("Interest is $%4.2f", interest);
```



`printf` Examples: Let's Practice

What is the formatted output for `double x = 23.45812` ?

```
System.out.printf("x is: %.2f", x);
```

```
System.out.printf("x is: %10.2f", x);
```

```
System.out.printf("x is: %.10f", x);
```

Math Functions

Math Functions : Using the `Math` class

A **method** is a group of statements that performs a specific task.

Java provides many useful methods in the **`Math`** class for performing common mathematical functions, such as:

- Square root
- Round up/down to nearest integer
- Generate a random number

Syntax for Calling a Method

```
object.method(arguments)
```

- object: Must have created the object first
- . (dot): Associates the method with the object
  - The method belonging to the object
- method: The action you want to do
- arguments: Extra info the method needs to work
  - Can have zero or more

Exponents & Square Roots

To raise variable x to the power of y (x^y): `pow(x, y)`

To find the square root of x: `sqrt(x)`

Example:

```
Math.pow(2,3)           // Output: 8.0
Math.pow(4.5, 2.5)      // Output: 42.95767
Math.sqrt(4)            // Output: 2.0
Math.sqrt(10.5)         // Output: 3.24037
```

Rounding Methods

Ceiling : round up to the nearest whole number: `ceil(x)`

Floor : round down to the nearest whole number: `floor(x)`

Example:

```
Math.ceil(2.1)          // Output: 3.0
Math.ceil(-2.1)         // Output: -2.0
Math.floor(2.1)         // Output: 2.0
Math.floor(-2.1)        // Output: -3.0
```

Minimum, Maximum, Absolute Value

Return the lower number given x and y: `min(x, y)`

Return the higher number given x and y: `max(x, y)`

Return the absolute value of a number: `abs(x)`

```
Math.min(2.5, 4.6)      // Output: 2.5
Math.max(2.5, 3)        // Output: 3.0
Math.abs(-2)            // Output: 2
Math.abs(-2.1)          // Output: 2.1
```

Random Numbers

Generate a random double value greater than or equal to 0.0 and less than 1.0: **`Math.random()`**

```
(int)(Math.random() * 10)          // 0 - 9
50 + (int)(Math.random() * 50)     // 50 - 99
```

Remember: **`x + Math.random() * y`** returns a random number between **`x`** and **`x + y`**, excluding **`x + y`**.

Strings

String Methods

`length()`	Return the number of characters in a string
`charAt(index)`	Return the character at the specified index
`concat(s1)`	Return a new string that concatenates with string s1
`toUpperCase()`	Return a new string with all letters in uppercase
`toLowerCase()`	Return a new string with all letters in lowercase

length() Example

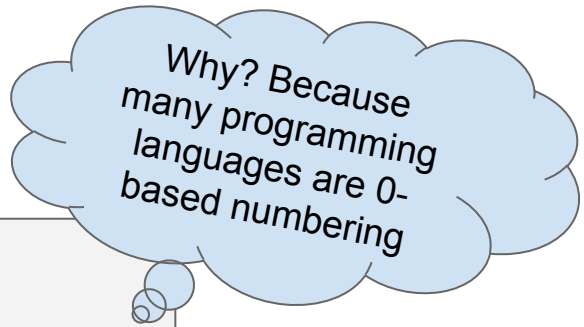
```
1  public class lengthExample {
2     public static main(String[] args) {
3
4        String welcome = "Welcome to Java";
5        System.out.print("The length of " + welcome +
6          " is " + welcome.length());
7
8     }
9  }
```

```
The length of Welcome to Java is 15
```

charAt(index) Example

```
1  public class charAtExample {
2     public static main(String[] args) {
3
4        String welcome = "Welcome to Java";
5        System.out.print("Character at index 0 is: " +
6          welcome.charAt(0));
7
8     }
9  }
```

```
Character at index 0 is: W
```



concat(someOtherString) Example

```
1  public class concatExample {
2     public static main(String[] args) {
3
4        String pb = "Peanut butter";
5        String andJ = " and jelly";
6        String pbAndJ = pb.concat(andJ);
7        System.out.print(pbAndJ);
8
9     }
10 }
```

```
Peanut butter and jelly
```

toUpperCase() Example

```
1  public class toUpperCaseExample {
2     public static main(String[] args) {
3
4        String hello = "Hello";
5        System.out.print(hello.toUpperCase());
6
7     }
8  }
```

```
HELLO
```

toLowerCase() Example

```
1  public class toLowerCaseExample {
2     public static main(String[] args) {
3
4        String hello = "Hello";
5        System.out.print(hello.toLowerCase());
6
7     }
8  }
```

```
hello
```

Handling Errors

Types of Errors

- Syntax Errors
- Runtime Errors
- Logic Errors

Syntax Errors (aka Compile Errors)

Syntax errors are from code construction errors:

- Mistyping a keyword (`string` instead of `String`)
- Forgetting necessary punctuation
- Using an opening brace but not a closing brace

Syntax Error Example

```
1  public class SyntaxErrorExample {
2     public static main(String[] args) {
3
4        System.out.print(“Welcome to Java);
5
6     }
7  }
```

Runtime Errors

Runtime errors cause a program to terminate abnormally (“crash”) because there is an operation that is impossible to carry out.

- Sometimes caused by input mistakes
  - ex. The program expects to read a number, but the user inputs a string.
- Division by zero

Runtime Error Example

```
1  public class RuntimeErrorExample {
2      public static void main(String[] args) {
3
4          System.out.print(1/0);
5
6      }
7  }
```

Logic Errors

Logic errors happen when a program doesn't perform the way it was intended to.

- Happen for all sorts of reasons!

Logic Error Example

```
1  public class LogicErrorExample {
2     public static void main(String[] args) {
3
4        System.out.print("35 Celsius in Fahrenheit:");
5        System.out.print((9/5) * 35 + 32);
6
7     }
8  }
```

```
 ---- jGrasp exec: LogicErrorExample
35 Celsius in Fahrenheit:
67
 ---- jGrasp: operation complete
```

Exceptions

Exceptions

An **exception** is an object that represents an error or a condition that prevents program execution from proceeding normally.

If the exception is not handled, the program will terminate abnormally -- aka crash!

- Crashing code is okay, but make sure you learn from it
- The end user (not you) should not have to experience crashes

Handling Exceptions

- Anticipate the possibility of exceptions happening
  - Even if you create very descriptive output labels for the user, they don't always read it
- Use Exception handling

Arithmetic Exceptions

– Example: divide by zero

– Every time that you will be performing division, check that the divisor is not zero

Know Your Operands

Division by zero may happen when you have the user enter numeric information

- User may enter in a zero
- Result of arithmetic in the code may result in a division by 0

```
1  public class ArithmeticErrorExample {
2     public static void main(String[] args) {
3
4        int num1 = 15;
5        int num2 = 7;
6        int num3 = 8
7        int dividend = 94;
8        int result = dividend / (num1 - num2 - num3);
9
10    }
11 }
```

```
 ---- jGrasp exec: divideByZeroExample
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at test.main(test.java:11)
```

Try-Catch

Try/Catch Syntax

```
try {

    // Risky code goes here
    // Can also have non-risky code
}
catch(exception and a name) {
    // When the exception happens, the code here will be
    // executed. Write a message to confirm!
}

// Program continues
```

The Catch Block

You should know the specific exception you are trying to handle.

How do you know the name?

- Java tells you when your program crashes!

```
 ---- jGrasp exec: DivideByZeroExample
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at test.main(test.java:11)
```

Catch Block Syntax

```
catch(exception and a name) {
    // When the exception happens, the code here will be
    // executed. Write a message to confirm that the error
    // was caught!
}
```

Example:

```
    catch(ArithmeticException ae) { }
```

```
1  public class ArithmeticErrorExample {
2     public static void main(String[] args) {
3
4        int num1 = 15;
5        int num2 = 7;
6        int num3 = 8;
7        int dividend = 94;
8        try {
9           int result = dividend / (num1 - num2 - num3);
10       }
11       catch(ArithmeticException ae) {
12          System.out.println("Error: Divison by 0 !");
13       }
14    }
15 }
```

`Scanner` Exceptions

Scanner exceptions happen when user enters incorrect input.

You will need to anticipate the types of errors the user can make.

Remember: Reading `int` Numbers

```
Scanner reader = new Scanner(System.in);

int userNum = 0;

System.out.print("Enter an integer: ");

userNum = reader.nextInt();

System.out.println();
System.out.print("The number is: ");
System.out.print(userNum);
```

Reading `int` Numbers – Error!

```
---- jGrasp exec: ScannerExceptionExample
Enter an integer: Ten
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Scanner.java:864)
        at java.util.Scanner.next(Scanner.java:1485)
        at java.util.Scanner.nextInt(Scanner.java:2117)
        at java.util.Scanner.nextInt(Scanner.java:2076)
        at scannerExceptionExample.main(scannerExceptionExample.java:27)

 ---- jGrasp wedge: exit code for process is 1.
 ---- jGrasp: operation complete.
```

Reading `int` Numbers – Let's Try Again!

```
Scanner reader = new Scanner(System.in);

int userNum = 0;

System.out.print("Enter an integer: ");

userNum = reader.nextInt();

System.out.println();
System.out.print("The number is: ");
System.out.print(userNum);
```

Identify the risky code.
Put it in **`try`**.

Reading `int` Numbers – Let's Try Again!

```
Scanner reader = new Scanner(System.in);

int userNum = 0;

try {
    System.out.print("Enter an integer: ");
    userNum = reader.nextInt();
    System.out.print("The number is: " + userNum);
}
catch(java.util.InputMismatchException ie) {
    System.out.println("That was not an int!");
}
```

`java.util.InputMismatchException`

When using the **`nextInt()`** method:

- If the user enters blanks before or after the int
- If the user enters a String that is NOT a number
- If the user enters a number with decimals

When using the **`nextDouble()`** method:

- If the user enters blanks before or after the double
- If the user enters a String that is NOT a number
- It WILL NOT crash if the user enters an integer number!

Using `nextLine()`

When using the **`nextLine()`** method:

- There are no exceptions when reading input
- Strings are characters, which can be numbers, letters, symbols, or empty string.

The `try` Block + Multiple Catch Statements

Should contain AT LEAST ONE risky statement inside.

If more than one exception may occur at runtime, use multiple catch statements:

```
try {
     // risky code
}
catch(ArithmeticException ae) {
     // catching an Arithmetic Exception
}
catch(java.util.InputMismatchException ime) {
     // catching an Input Mismatch Exception
}
```

```
// Create a Scanner object -- so we can read the user's input
Scanner reader = new Scanner(System.in);

int userNum = 0;    // Create a variable to store the input

try { // risky! What if user puts in a string, double, or 0?!
    System.out.print("Enter a non-zero integer: ");
    userNum = reader.nextInt();
    System.out.print("100 divided by your # is:" + 100/userNum);
}
catch(ArithmeticException ae) {          // possible error: 100/0
    System.out.println("Error: Non-zero integers only please!");
}
catch(java.util.InputMismatchException ime) {    // non-int input
    System.out.println("Error: That was not an int!");
}
```

The `finally` Block

The **`finally`** block is placed after all catch blocks.

It will always execute even if exceptions are caught.
If no exceptions are caught, it will execute after the **`try`** block.

You should place those statements that must execute whether exception occurs or not into the **`finally`** block.

Try/Catch/Finally Syntax

```
try {

    // Risky code goes here (non-risky code is ok too)
}
catch(exception and a name) {
    // When the exception happens, the code here will be
    // executed. Write a message to confirm!
}
finally {
    // Code here will always execute if the try block exists
}
```

Strings & Exceptions

The “Nice” String Methods

These will not crash your code:

- concat
- toUpperCase
- toLowerCase
- length

The “Not Nice” String Methods

These may crash your code:

- charAt

`charAt(int position)` Exceptions

Remember: **`charAt()`** returns the character given a position.

Exceptions will happen if the position is out of range!

- If the position is negative
- If the position is greater or equal to the length of the string
  - Remember: string of size n has positions from 0 to n - 1

```
1  public class CharAtErrorExample {
2     public static void main(String[] args) {
3
4        String hello = "Hello!";
5        char tenthCharacter = hello.charAt(10);
6
7
8     }
9  }
```


```
 ---- jGrasp exec: CharAtErrorExample
Exception in thread "main" java.lang.StringIndexOutOfBoundsException: String
index out of range: 10
        at java.lang.String.charAt(String.java:695)
        at charAtErrorExample.main(charAtErrorExample.java:17)
```

Try/Catch with `charAt()`

```
try {

    // Risky code using charAt()
}
catch(StringIndexOutOfBoundsException se) {
    // Message to the user about why the program can't
    // do what they are trying to do.
}
```