CSCI 1103: Input with TextIO, Basic Types

Chris Kauffman

Last Updated:
Wed Sep 20 21:38:55 CDT 2017

Logistics

Reading

Eck Ch 2

- Available online: http://math.hws.edu/javanotes/
- Reading ahead is encouraged

Goals

- Input from user
- Variable Types
- Arithmetic Stuff
- Methods

Project 1

- Will be posted by Friday, discuss then
- Due end of next weekend
- 2-3 short programs

Exercise: Quick Review

- What's a *variable*?
- Draw a picture of the MEMORY layout of the following java program:

```
  int young = 10;
  int old = 98;
  int diff;
  young = young + 1;
  diff = old - young;
> // draw program at this point
```

- How would one print the value of `diff` on the SCREEN?
- What *incantations* must be added to get the above to actually run?

Birthday Exercise

Write/review the Birthday java program (end of previous lecture).

Compile Time vs. Runtime

Compile Time: Translate `X.java` to `X.class`

- Java compiler translates high level, human-readable Java code to low-level, machine readable bytecode
- `X.java` source file compiles to `X.class` bytecode/class file

```
> ls                          # show what's in this folder
X.java                        # 1 file: X.java
> file X.java                 # what kind of file is X.java
X.java: ASCII text
> javac X.java                # compile X.java
> ls                          # show what's in this folder
X.class  X.java               # 2 files: X.java and X.class
> file X.class                # what kind of file is X.class
X.class: compiled Java class data, version 52.0 (Java 1.8)
```

Runtime

- A compiled Java program is loaded executed by the CPU
- Given memory boxes, print stuff to screen
- After making changes to `X.java`, must re-compile to see the changes when it runs - DrJava is aware

Dynamic Input for Programs

- Changing variables and re-compiling every time is a drag
  - My age is 36
  - Re-edit `Birthday.java` to set `int age=36;`
  - Re-compile, re-run
  - Deanna's age is 19
  - Re-Edit `Birthday.java` to set `int age=19;`
  - Re-compile, re-run
  - Amy's age is 30
  - . . . *someone kill me now. . .*
  - NO: Just re-write to ask for age
- Frequently programs must get input from somewhere
- Easiest input to understand is directly from user of program
- Will allow program to have different behavior based on different input

Input In Java

- Input in Java is a pain due to early decisions in Java
- We will use the Eck's textbook approach `TextIO.java`
- Make sure that `TextIO.java` is present in the same folder as your other programs (make copies if needed)
- Provides a simple way to get input from users

```
int age = TextIO.getInt();
```

- Input is often preceded by a prompt describing what's happening

```
System.out.println("Enter your age:");
int age = TextIO.getInt();
System.out.println("I hear you are " + age);
```

Podunk Model and Input

Input is a little hard to write on SCREEN in examples but with prompts, context should resolve ambiguities

```
CPU: at instruction 10:                    MEMORY:            SCREEN:
> 10: println("Enter your age");           | Name | Value |
  11: int age = TextIO.getInt();           +------+-------|
  12: println("I hear you are " + age);    | age  |     0 |

CPU: at instruction 11:                    MEMORY:            SCREEN:
  10: println("Enter your age");           | Name | Value |  Enter your age:
> 11: int age = TextIO.getInt();           +------+-------|
  12: println("I hear you are " + age);    | age  |     0 |

CPU: at instruction 12:                    MEMORY:            SCREEN:
  10: println("Enter your age");           | Name | Value |  Enter your age:
  11: int age = TextIO.getInt();           +------+-------|  22
> 12: println("I hear you are " + age);    | age  |    22 |

CPU: at instruction 12:                    MEMORY:            SCREEN:
  10: println("Enter your age");           | Name | Value |  Enter your age:
  11: int age = TextIO.getInt();           +------+-------|  22
  12: println("I hear you are " + age);    | age  |    22 |  I hear you are 22
> 13: ...
```

Question: Why is age initially 0 at the beginning?

Exercise: FruitStand

Pseudocode

- Prompt for `apples`, read integer
- Prompt for `oranges`, read integer
- Print for `apples`
- Print for `oranges`
- Print total `fruits`

Draw a MEMORY diagram of the running programc

Use

```
public class FruitStand{
  public static void main(String args[]){

    System.out.println("stuff");
    int x = TextIO.getInt();
```

Sample Session

```
> javac FruitStand.java
> java FruitStand
How many apples?
1
How many oranges?
2
apples: 1
oranges: 2
fruits: 3
> java FruitStand
How many apples?
800
How many oranges?
303
apples: 800
oranges: 303
fruits: 1103
```

Answer: FruitStand

```
public class FruitStand{
  public static void main(String args[]){

    System.out.println("How many apples?");
    int apples = TextIO.getInt();
    System.out.println("How many oranges?");
    int oranges = TextIO.getInt();
    int total = apples+oranges;

    System.out.println("apples: " +apples);
    System.out.println("oranges: "+oranges);
    System.out.println("fruits: "+total);
  }
}
```

Note: `TextIO` or Not?

- `TextIO` is available from the Textbook but may not be available every time you use Java
- Common alternative is the `Scanner` class which is a bit more complex
- We will use `Scanner` later in the class
- For now `TextIO` is simple and slick

Other Primitive Variable Types

While useful, `int` is not the only game in town. Here are ALL of Java's primitive types

1103	Name	Bytes	Range
	byte	1	-128 to 127
X	int	4	-2,147,483,648 to 2,147,483, 647
	short	2	-32,768 to 32,767
	long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
	float	4	±3.40282347E+38F (6-7 significant decimal digits)
X	double	8	±1.79769313486231570E+308 (15 significant decimal digits)
	char	2	0 to 65,536 (unsigned)
X	boolean	2(?)	`true` or `false`
X	reference	4 / 8	Pointer to another memory location, 32 or 64bit

A primitive type fits in a single memory box with the given size

```
                              MEMORY
int a = 5;                  | #1024 | a |     5 |
double x = 1.23;            | #1028 | x | 1.23  |
boolean b = true;           | #1036 | b | true  |
int c = 2;                  | #1038 | c |     2 |
```

Exercise: Draw a MEMORY Diagram

1103	Name	Bytes	Range
X	int	4	-2,147,483,648 to 2,147,483, 647
X	double	8	±1.79769313486231570E+308 (15 significant decimal digits)
X	boolean	2(?)	`true` or `false`

- Draw a memory diagram of the following variables.
- Make sure that the memory addresses of the boxes reflect the sizes in bytes of the types given

```
double x = 1.23;
double y = 4.56;
int myInt = 15;
boolean bool = false;
double z = 4.56;
boolean bool2 = true;
```

Exercise: Number operations: `int` and `double`

Arithmetic operations for both `int` and `double`

+	addition	*	multiplication
-	subtraction	/	division (!)

Generally can mix arithmetic of `int` and `double`, but some gotchyas exist for division:

```
double x = 10.0;
double y = 3.0;

double z = x / y;
// What is z?

double w = a / b;
// What is w?

double r = a / (double) b;  // Casting
double t = (double) a / b;  // Casting 2
```

```
int a = 10;
int b = 3;

int c = a / b;
// What is c?

double u = a / y;
// What is u?
```

- Verify these in the interactive loop if DrJava
- Understand WHY each result happens
- Take care when mixing integral and floating types
- Arithmetic can be complex:

  `x = (x + S/x) / 2.0;`

Division for `int`

- `int q = a / b;` means divide and get the quotient
  - how many times does `b` "go into" `a`)
- `int r = a % b;` means divide and get the remainder
  - What's left from `b*q - a`

The symbol % (percent) is often referred to as the modulo operator

- Works ONLY for integers
- No remainder for `double`: leftovers become fractions

Note: there are a bunch of other things that can be done with `ints`, bitwise operations, that we may deal with later in class. These have symbols like <<

Logical operations: `boolean`

The `boolean` type represents either `true` or `false` as in

```
boolean a = true;
boolean b = false;
```

Booleans have a set of logical operators which manipulate them.

```
boolean x = a && b; // logical AND: true only if both a,b are true
boolean y = a || b; // logical OR:  false only if both a,b are false
boolean z = !a;     // logical NOT: flips true to false, false to true
```

These can be combined in similar ways to arithmetic.

```
boolean w = !(a && b);
boolean t = !w || (!b && a);
```

- Values for the above booleans?
- `boolean` types get more action in control structures

Exercise: Reading Data

`TextIO` provides easy facilities to ask for basic types

`int i = TextIO.getInt();`	Read an integer from the user
`double x = TextIO.getDouble();`	Read an double from the user
`boolean b = TextIO.getBoolean();`	Read an boolean from the user

Identify in each situation which of these to use
Need to know. . . .

- if user is a student or not
- GPA of user
- the age of user
- how much cash they have in their pocket
- credit card number
- which major they pick. . .

Math Methods

- Arithmetic is available via symbols: `+,-,*,/`
- More complex operations come from the `Math` class
- `System` allows printing via `System.out.print()`
- `Math` is similar but has math operations

```
double rootOfTwo = Math.sqrt(2.0);
// 1.4142135623730951

double fiveToPower = Math.pow(5.0, 7.3);
// 126613.79661662203

double x = 7.8;
double y = 2.3;
double xToY = Math.pow(x,y);
// 112.67241063690722
```

Full listing of `Math` operations is in the Java Doc:

https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

Exercise: Math!

Use the `Math` class functions

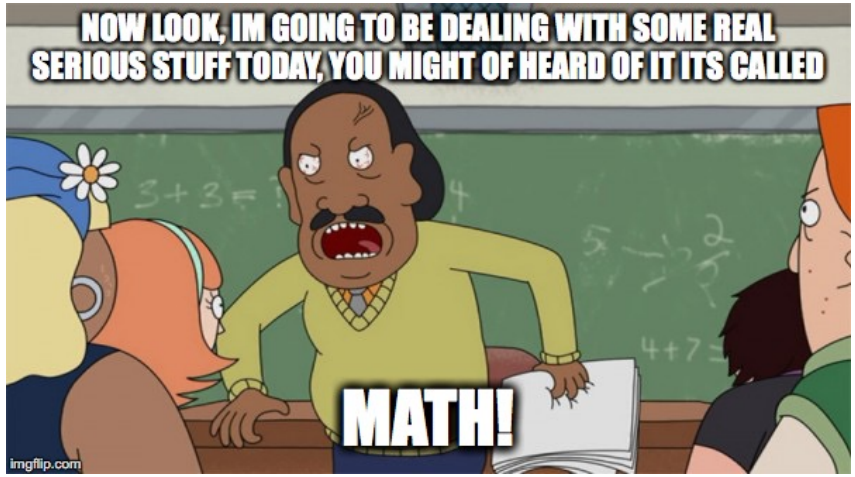
- `Math.sqrt( z )`
- `Math.pow( m , n)`

to compute the following two values, `x` and `p`.

$$x = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

```
double a = 3.5;
double b = -4.1;
double c = 0.5;
double numerator   = ???;
double denominator = ???;
double x = ???;
```

What was this thing again?



$$P = Q \times e^{r \times t}$$

```
double q = 25.0;
double e = 2.718;
double r = 2.0;
double t = 1.7;
double p = ???;
```

Anyone familiar with this gem?

Solution: Math!

```
// Solution to in-class exercises on using Math.sqrt() and Math.pow()
public class DoMath{
  public static void main(String args[]){
    double a = 3.5;
    double b = -4.1;
    double c = 0.5;
    double numerator   = -b + Math.sqrt(b*b - 4*a*c);
    double denominator = 2*a;
    double x = numerator / denominator;
    System.out.println(x);

    double q = 25.0;
    double e = 2.718;
    double r = 2.0;
    double t = 1.7;
    double p = q * Math.pow(e, r*t);
    System.out.println(p);
  }
}
```

Printing formatted output

- `System.out.println(myDouble)`, easy to call, prints lots of digits
- `System.out.printf(..)`: more complex, more control over numbers

```
double x = 1.2345678912345678 9;
System.out.println(x);
// 1.234567891234568

System.out.printf("%.4f\n",x);
// 1.2346
```

- `System.out.printf( format, arguments...)`: takes 2 arguments
  - `format` controls how things will be printed, is a `String`
  - `arguments..` are things to substitute into the format
- `"%.4f\n"`
  - Substitutions start with a % sign
  - `.4` means 4 decimal digits
  - `f` means floating point number
  - `\n` means "new line"
- `System.out.printf("%.4f\n" , x);`
  - Print `x` as a floating point number with 4 digits of accuracy followed by a newline

Recipes for `printf()`

```
double x = 1.23456789;
double y = 4.95;
double z = 0.00789;

// print only x with 2 digits
System.out.printf("x is %.2f\n",x);
// x is 1.23

// print x,y,z with 2 digits
System.out.printf("all are %.2f %.2f %.2f\n",x,y,z);
// all are 1.23 4.95 0.01

// print x,y,z with 3 digits
System.out.printf("3 digs %.3f %.3f %.3f\n",x,y,z);
// 3 digs 1.235 4.950 0.008

// mixed precision
System.out.printf("x: %.5f... y: $%.3f z: %.0f\n",x,y,z);
// x: 1.23457... y: $4.950 z: 0
```

Notice `printf()`

- Does rounding automatically
- Can handle multiple substitutions
- Can include literal text like $ (project 1)

Exercise: printf()

```
double x = 1.0331559932390235;
double q = 748.8384692277563;

// Use a single printf() to print x to 5 decimal
// digits and q to 2 decimal digits. Include a $
// sign before q and a newline at the end.

System.out.printf(????);

// x: 1.03316  y: $748.84
```

String Data

```
String name = "Chris";
String occupation = "csci prof";
String university = TextIO.getWord(); // enter: UMN
```

- A class with specific instances which are objects
- Also called a reference type
- Strings are fundamentally different than the primitive types
- Simplified memory picture: what should be at address #4000

```
| #1024 | name       | #2048 |      | #3032 | [0] | 'c' |
| #1028 | occupation | #3032 |      | #3034 | [1] | 's' |
| #1032 | university | #4000 |      | #3036 | [2] | 'c' |
| ...   | ...        | ...   |      | #3038 | [3] | 'i' |
| #2048 | [0]        | 'C'   |      | #3040 | [4] | ' ' |
| #2050 | [1]        | 'h'   |      | #3042 | [5] | 'p' |
| #2052 | [2]        | 'r'   |      | #3044 | [6] | 'r' |
| #2054 | [3]        | 'i'   |      | #3046 | [7] | 'o' |
| #2056 | [4]        | 's'   |      | #3048 | [8] | 'f' |
| ...   | ...        | ...   |      | ...   | ... | ... |
```

Primitives and References

Primitives

- There are about 8 primitive types in Java like `int`
- You cannot create new primitive types
- All of them start with lower case letters: `double, boolean`
- Values of primitives fit entirely inside their memory box
- Primitives have no methods: can't do anything

Reference types

- There are tons of reference types
- You will create many more: `public class MyType{`
- They start with upper case letters: `String, Scanner`
- A variable with a reference type has a memory box but it's contents refers to another spot in memory
- Reference types typically have methods: can do things

String Method Examples

```
String name       = "Chris";
//                   01234
String occupation = "csci prof";
//                   012345678
// Example Methods
int nameLength = name.length();       // ask for the length of name
int occLength = occupation.length(); // length of occupation
char third = name.charAt(3);          // third character of "Chris"
char fifth = occupation.charAt(5);    // third character of "csci prof"
String subString = name.substring(1,4); // "hri" chars 1 to 3
String changed = occupation.replace("prof","badass"); // smirk
```

- Strings have many methods
- Complete list is in the Java documentation:

https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

Exercise: Name Length

- Prompt a user for their name
- Calculate length with `str.length()`
- Print out the number of characters in the name

```
> javac NameLength.java

> java NameLength
What's your name?
Amy
Amy: did you know your name has 3 characters?

> java NameLength
What's your name?
Christopher
Christopher: did you know your name has 11 characters?

> java NameLength
What's your name?
Professor Kauffman
Professor: did you know your name has 9 characters?
```

Note the last run measured only the 9 characters in Professor

Answer: Name Length

```
// Solution to name length exercise
public class NameLength{
  public static void main(String args[]){
    System.out.println("What's your name?");
    String name = TextIO.getWord();
    int length = name.length();
    System.out.println(name+": did you know your name has "+
                       length+" characters?");

// ALTERNATIVE: print with printf()
// System.out.printf("%s: did you know your name is %d characters?\n",
//                   name, length);
  }
}
```

- Notice that it is fine to break of the long `println()` call across several lines: compiler doesn't care and humans can read easier won't matter
- The alternative uses `printf()` with `%s` to sub strings and `%d` for integers