CS 211
Control Flow
Arrays

More Java Basics

Strings, Basic I/O

Control Flow

Arrays

Strings and I/O

Strings

- String literals must be surrounded by double-quotes
- There is no multi-line string.
- Escape characters exist. e.g. `\n` `\t` `\'` `\"` `\\`
- Add Strings together with `+`, e.g. `"Peanut "+"butter"`
- Note: + can also add anything to a String to get a String!
  → The + operator is also used for arithmetic addition
  → operand types dictate which meaning + has.
  → operands are evaluated left to right, but parentheses can drive meaning:

  `5+6+"a"` vs `5+(6+"a")`

Printing

- We can call the following methods to send characters to the screen.

- Print String and a newline character:
  `System.out.println(stringExpr)`

- Print String, and no extra newline character:
  `System.out.print(stringExpr)`

- plug substitutions into the format, print it out:
  `System.out.printf(formatExpr, substs … )`

Reading Input

- We can get String inputs from the user
- The `Scanner` class is a good interface choice
  - a Scanner object can be attached to various sources, like the keyboard, a file, a String, or other places.
- The keyboard's input is represented by the `System.in` object

Reading Input

The following line creates a `Scanner` object that reads from the keyboard:

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

The `new` operator creates the `Scanner` object

Once created, the `Scanner` object can be used to invoke various input methods, such as:

```
// read to end of line
answer = scan.nextLine();
```

Reading Input

The `Scanner` class is part of the `java.util` class library, and must be imported into a program to be used. (add import java.util.Scanner; at top of file)

The `nextLine` method reads all of the input until the end of the line is found

The `System` class

Refers to the operating system, which handles input/output for programs you write

- `System.out`
- `System.in`
- `System.err`

These are all *buffers* you have access to from the System class

Input Tokens

Unless specified otherwise, whitespace separates all other characters into "tokens", and we can read one at a time.

The **`next`** method of the **`Scanner`** class reads the next input token and returns it as a string

nextInt reads the next token and converts it to an int (this could fail)

Comparing Strings

String literals still become objects of the String class.

The `equals` method can be called with Strings to determine if two Strings contain exactly the same characters in the same order.

The `equals` methods returns a boolean.

Example:

```
if (name1.equals(name2)) {
        System.out.println("jinx!");
}
```

Escape Sequences

Some Java escape sequences:

Escape Sequence	Meaning
`\b`	backspace
`\t`	tab
`\n`	newline
`\r`	carriage return
`\"`	double quote
`\'`	single quote
`\\`	backslash

Escape Sequence: Example

```
System.out.println ("Roses are red,\n\tViolets are blue,\n" +
        "Sugar is sweet,\n\tBut I have \"commitment issues\",\n\t" +
        "So I'd rather just be friends\n\tAt this point in our " +
        "relationsha\bip.");
```

output:

```
Roses are red,
      Violets are blue,
Sugar is sweet,
      But I have "commitment issues",
      So I'd rather just be friends
      At this point in our relationship.
```

Pytania Poll

- Strings and I/O

Constants

constant: a place to store a value, which may not be changed. It's the "no reassignments allowed" variable.

The compiler will issue an error if you try to change the value of a constant

In Java, we must use the `final` modifier to declare a constant (and CAPS_WORDS names are the common naming convention):

```
final int MIN_HEIGHT = 60;
```

Increment/Decrement

Shorthand allows us to increment or decrement a number:

```
x++   ++x   //increment     (after/before stmt)
x--   --x   //decrement   (after/before stmt)
```

Often these are one-liners or isolated:

```
x++;

for (int i=0; i<10; i++) {
```

Increment/Decrement

These are expressions, too:

Suffix form (**x++**, **x––**) : use the current value in the enclosing statement, then inc./dec. this variable **after**.

```
int x=1;                              int x=1;
int y = (x++) * 5;        →           int y = x * 5;
                                      x = x + 1;
```

Prefix form(**++x**, **––x**): perform inc./dec. **before**, using the new value in the enclosing statement.

```
int x=1;                              int x=1;
int y = (++x) * 5;        →           x = x + 1;
                                      int y = x * 5;
```

Control Flow

Control Flow

- Boolean expressions
- if / if-else
- switch
- while, do-while
- for (original and Iterator versions)
- break, continue

→ Try each structure out in code as we explore them. If we're not coding, we're not 'learning to program': we're only 'learning about programming'.

Boolean Expressions

Control flow uses **boolean expressions** to navigate blocks of code.

How do we get booleans?

- directly, with **true** and **false**
- using relational operators: **<   <=   >   >=   ==   !=**
- using boolean operators: && || !
- calling a method that returns a boolean
  e.g. **myScanner.hasNext()**
- any expression, as long as it results in **true** or**false**

Block Statement

- Multiple statements can be grouped into a single "compound statement" with curly braces **{ }** . Example:

```
{
      stmt1;
      stmt2;
      ...
}
```

- It's so common with control structures that it seems like {}'s are part of their syntax, but it is a separate statement structure all on its own.

If-statement

Syntax:

```
if ( boolexpr )
     stmt
```

Semantics:

evaluate boolexpr. If it was true, evaluate stmt. If it was false, skip stmt.

Examples:

```
if (x>100)
  System.out.println("x is big!");


if (y<10) {
  System.out.println("y is too small.");
}
```

If-Else Statement

Syntax:

```
if (boolexpr)
      stmt1
else
      stmt2
```

Semantics:

evaluate boolexpr. If it was true, only evaluate stmt1. If it was false, only evaluate stmt2. (Note exactly one of stmt1 and stmt2 always runs). Example:

```
if ( dist >0.8*au  && dist<1.5*au )
  System.out.println("planet may be habitable!");
else
  System.out.println("probably ice cube/plasma.");
```

'Else if' in Java

There is no 'elif' in Java: just chain "if else" statements together:

```
if       (be1) s1
else if (be2) s2
else if (be3) s3
else s4
```

=

```
if (be1) {s1}
else {
    if (be2) {s2}
    else {
        if (be3){s3}
        else s4
    }
}
```

- exactly one of s1, s2, s3, and s4 runs each time (corresponding to which boolexpr is found true first, visited in order)
- `if` and `else` grab one statement to their right
- precedence can always sort out which branch belongs where.
- The final "else" branch is still optional: innermost if-else replaced with if-statement. (In this described case, at most one of s1, s2, s3 runs).

Switch Statement

Syntax:

```
switch (expr) {
    case val1: stmt1
    case val2: stmt2
    …
    default: stmtD  // 'default' case optional
}
```

Semantics:

- expr must be integral (whole number), char, String, or enum
- All case values must be constants, and same type as expr.
- evaluate expr, enter {}'s at matching case (or default)
- execute ***all*** stmts after matching case!
  - `break` is common at the end of each case

Switch Statement Example

```
Scanner sc = new Scanner
(System.in);
int x = sc.nextInt();
int v = 0;
switch (x) {
      case 1:
            v = 1;
            break;
      case 2:
            v = 20;  //note: no break!
      case 3:
            v = v + 3;
            break;
      case 4: case 5: case 6:
            v = 456;
            break;
      default:
            v = 999;
}
```

Input:	v value:
0	999
1	1
2	**23**
3	3
4	456
5	456
6	456
7+	999

(w/o **default:** 0, 7+:
v exhibits no change)

Practice Problems

- Convert the previous slide's switch statement to an if-else structure.
- What would make a series of if-else statements a good candidate for a switch statement?
- What are the limitations? Reasons to choose?

Pytania Poll

- Selection Statements

While Loop

Syntax:

```
while (boolexpr)
    stmt
```

Semantics:

- evaluate boolexpr.
  → true? execute stmt and retry. → false? exit loop.
- if boolexpr is false on first time, stmt is never run!
- if stmt can't make boolexpr false, the loop is infinite.

Example:

```
while (x<100) {
    System.out.println(x);
    x = x+1;
}
```

Do-While Loop

Syntax:

```
do stmt
while (boolexpr);
```

Semantics:

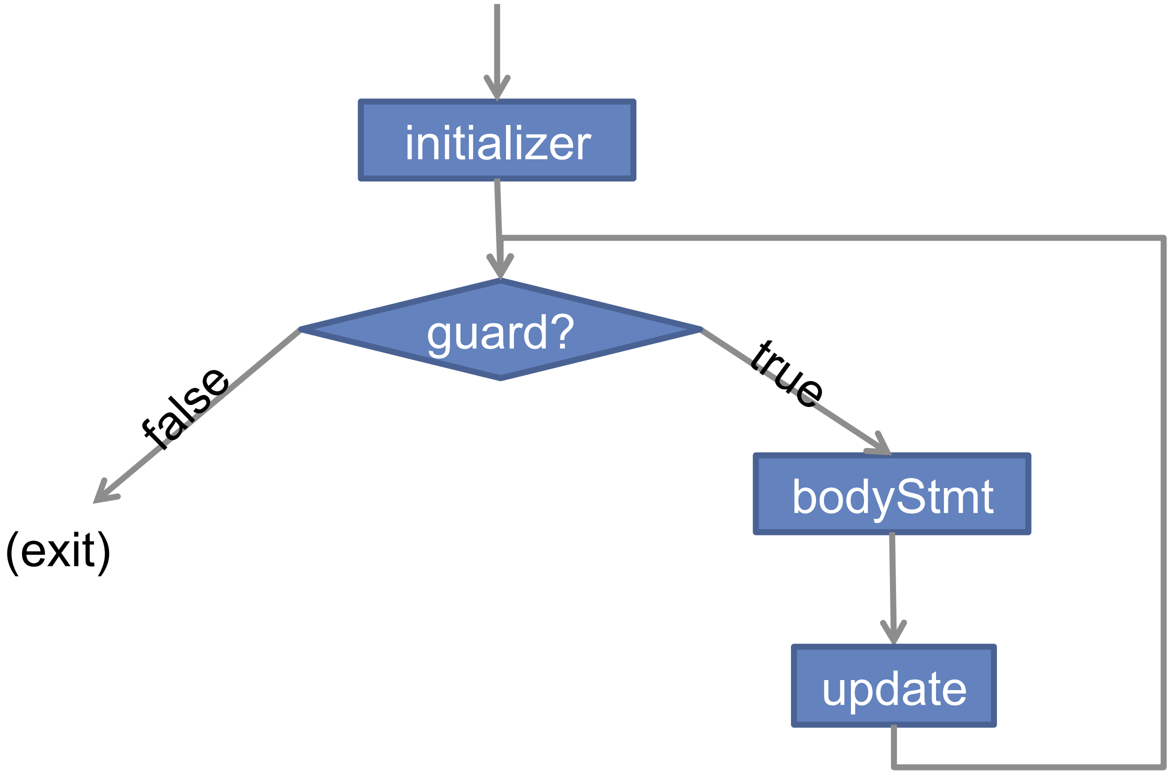
- evaluate stmt (no matter what).
- evaluate boolexpr; → true? repeat. → false? exit loop.
- semicolon after (boolexpr) is required! `;`
- Note: stmt runs at least once

Example:

```
int x = 0;   //consider also x = 500;
do
  System.out.println(x++);
while (x<100);
```

for loop

```
for (initializer ; guard ; update)
    bodyStmt
```



- initializer may declare variable (scoped to loop) or use existing variable.
- can omit any of initializer/guard/update! Valid: *`for(;;) bodyStmt`*

Understanding the For Loop

The following two pieces of code would run **identically**: (other than if init declares a variable that only exists inside the loop):

```
for (init; guard; update) {
        stmt;
}
```

```
init;
while (guard) {
        stmt;
        update;
}
```

Common Pattern vs Python

index-loops from Java and Python:

```
//Java
for (int i=0; i<xs.length; i+=1){
    System.out.println(xs[i]);
}
```

```
#Python

for i in range(0, len(xs), 1):
    print(xs[i])
```

Practice Problems

- Use a for loop to print the numbers 1-1000 on the screen.

- Use a for loop to calculate the sum of the first 100 numbers, and then print it once to the screen.

- Without using an if-statement, use a for loop to print the numbers 100, 95, 90, 85, …,60 to the screen.

for-each Loop

Any "iterator" (including arrays) may be used to access one value at a time:

Syntax: `for (Type identifier : iteratorExpr) stmt`

```
for (Type identifier : iteratorExpr)
         stmt
```

Semantics:

- for each item in iteratorExpr, in order:
- assign the value to identifier; run stmt.
- Example:

```
int[] vals = {2,4,6,8};        // an array
for (int v : vals)
    System.out.println("seeing "+v);
```

Comparing to Python

for-each loops in Java and Python:

```
// Java
for (int x : xs){
    System.out.println(x);
}

# Python
for x in xs:
    print(x)
```

Other Control Flow Options

Some other control flow statements:

- `break` (immediately leave nearest loop)
- `continue` (immediately skip to next iteration of loop)
- `return` (immediately exit a method)

Arrays

Array Types

- The array type is indicated with **[ ]**'s.
- **Monomorphism:** Just as variables can only hold one type of value, Java arrays can only hold one specified type of value, in every slot.
- Example array types:

  ```
  int[]    double[]   boolean[][]      Person[]
  ```

- The type doesn't record the dimension lengths, but an array value will specify the (unchanging) lengths.

  ```
  //a 3x4 structure of ints.
  int[][] xs = new int[3][4];
  ```

Declaring an Array

We **declare** an array as a variable with an array-type :

- `int[] nums;                // an array of int values`
- `double[] scores;           //an array of double values`

Multiple dimensions can be 'stacked' together

- `short[][] twoDims;   //a 2D array of short values.`
- `float[][][] space;   //a 3D array of float values.`

Java arrays must entirely have the same ***number*** of dimensions.

- each 'row' of a dimension will be same type, e.g. `int[]`
- same-array values don't actually need to be the same length.

Creating Array Values

- at declaration: explicit listing of values

```
int[]      xs = {2,5,3,6,4} ;
double[][] ys = {{1.0,2.2}, {0.3,4}, {7.7,8.9}};  //3x2 dim.
```

- using the **new** keyword and specifying dimensions:

```
short[]    xs = new short[10];      //holds 10 shorts.
double[][] ys = new double[10][15]; //holds 10x15 doubles.
```

- anywhere, with full type in front of it:

```
new int[]{1,2,3}
```

- The length of each value in a multi-dimensional array may vary:

```
int[][] zs = {{0},{1,2,3,4,5},{6,7}};
int[][][] ws = {{{0},{1}},{{2,3,4,5,6}}};
```

Accessing/Modifying Arrays

- indexing via brackets **[ ]**:

```
a = xs[4];      //accesses 5th elt. of xs.
xs[0] = 7;      //replaces 1st elt. of xs with 7
```

- Any expression of type **`int`** may be used as an index, *regardless of the type in the array*:

```
xs[ a+4 ]       xs [ sc.nextInt() ]
xs[ i ]         ys [ i ][ j ]
```

- The length of an array is available as an attribute:

```
xs.length       ys[i].length
```

Arrays And Loops

BFF's Forever

Traditional index-style loop:

```
for (int i=0; i<xs.length; i++){
    … xs[i] …
}
```

Newer for-each-style loop:

```
for (int x : xs) {
    … x …
}
```

Practice Problems

- Use an array and a loop to find the maximum *value* in the array. (Give the array starting values).

- Use an array/loop to find the **index** of the maximum value in the array. (Give the array starting values).

- Sum every third value in the array, starting with the value at position 0.

Arrays vs lists

An array is not the same as a list (e.g., Python lists)

- Array: length permanently determined at creation. Fast access to all locations.
- List: length may vary over time. Slower access.
- Lists are often implemented via intelligent use of arrays to regain some of the speed of access without losing the ease of usage.

Pytania Poll

- Arrays

Exceptions

Exceptions: The Idea

exceptional events may occur during program execution.

- array index is out of bounds
- we divided by zero
- we tried to open a non-existing file
- many others…

Normal sequential control flow is aborted, in search of a way to handle the exceptional event.

- keep escaping code blocks until one is found.
- escaping out of main crashes whole program

Exception Types: A Class Hierarchy

- A **small portion** of the huge class hierarchy of Exceptions already defined in Java.
- You've perhaps already seen a few of these.

- **java.lang.Throwable** *(implicitly inherits from Object).*
    - **Exception**
        - **RuntimeException** *for recoverable events.*
            - **NullPointerException** *tried using null like an object*
            - **ClassCastException** *cast to class-type that wasn't possible*
            - **IndexOutOfBoundsException**
                - **ArrayIndexOutOfBoundsException**
                - **StringIndexOutOfBoundsException**
            - **ArithmeticException** *bad arithmetic, like "divide by zero"*
        - **IOException**
            - **FileNotFoundException** *attempted to open non-existing file*
            - **EOFException** *end of file reached (no more content)*
    - **Error** *unrecoverable events: e.g., out of memory*

Getting/Creating Exception Values

basic Java expression usage: Some Exception values are created through incorrect value usage. Examples:

- **dividing by zero** will cause an `ArithmeticException`
- using an **out-of-bounds index** will cause an `ArrayIndexOutOfBoundsException`
- `FileNotFoundException` thrown by FileInputStream constructor when the **file is not found**. (Any method might cause an exception)

creating your own: You can create your own:

- call the constructor of an Exception class. You'll also need to 'throw' it:

```
ArithmeticException ae = new ArithmeticException("evens only!");
throw ae;
```

Catching Exceptions: `try-catch` Blocks

Wrap the suspicious code in a `try`-block.

Provide a way to handle the occurring exception with a `catch`-block. This must include the type of Exception being caught.

→ if the exception occurs in the try-block, the catch block runs.

```
try {
   int infinity = 5 / 0 ;
   System.out.println("I'm never printed. " + infinity);
}
catch (ArithmeticException e) {
   System.out.println("saw arith. error: " + e);
}
```

Handling Exceptions

Catch It: wrap the offending code in a `try-catch` block that catches the specific type of exception.

Defer It: allow the exception to occur, propagating ('crashing') its way through your program until it is caught elsewhere.

- might have to explicitly list what exceptions are deferred (any that aren't a **`RuntimeException`** or **`Error`**).

No matter what, the occurring exception immediately starts 'crashing' your program by prematurely leaving each code block and method call, until it is caught by a catch-block (or the entire program is crashed).

Practice Problems

- Write code using a `try-catch` block that successfully gets an integer from the user, using a `Scanner`.

- use a `try-catch` block that converts a user's String input to an int using the parseInt method
  - can be called as **Integer.parseInt(someStringExpr)**
  - Return -1 if the parsing fails. (What Exception to catch?)

Pytania Poll

- Exceptions