Object Oriented Programming with Java

Control Flow

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

- As we introduce blocks of code for the branches/loops of control flow, we want to group many statements together.
- In Java, we place curly braces **{ }** around multiple statements to group them.
- It is so common to use them with control structures that it seems like {}'s are part of their syntax, but it is a separate statement structure all on its own (creates a ***scope*** too).

Example:

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

Control Flow Options

- if / if - else / if - else if - else if - … - else
- while
- do-while
- for (initialization; condition; update)   // for loop
- for (variable : iterable)   // foreach loop
- switch
- break
- continue

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 might be habitable!");
else
      System.out.println("probably an icy or molten blob.");
```

“elseif” in Java

There is no “elif” or “elseif” in Java, just a chain of "if else"

```
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
    }
}
```

- Java's `if` and `else` grab one statement to their right, so precedence can always sort out which branch belongs where.
- exactly one of s1, s2, s3, and s4 runs each time (corresponding to which boolexpr is found true first, visited in order)
- 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' is optional
}
```

Semantics:

- expr must be integral (whole number) or char, or String (no booleans or floats or objects!). All case values must be constants.
- evaluate expr, and compare against each case value in order until exact match is found.
- execute **all statements after** matching case! (thus `break` is common at the end of each case)
- default: no value; stmtD always runs if no other case values equaled the switch expression.

Switch Statement Example

```
Scanner sc = new Scanner(System.in);
int v=0, x=sc.nextInt();
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;
}
```

User Input:	Resulting v value:
0	999
1	1
2	23
3	3
4	456
5	456
6	456
7 (and up)	999

?

* without a **default:** 0, 7-and-up would exhibit no change to v

Switch Statement Example

```java
final int i = 5;
int y = 15;
final int z;
z = 25;
int x = 10;

switch(x)
{
    case i:  // OK: i can't be changed at any point due to the `final` modifier
        break;

    case y:  // ERROR: y isn't a compile-time constant
        break;

    case 10+10:  // OK: 10+10 is a compile-time literal that won’t change
        break;

    case z: // ERROR: wasn't initialized with declaration, isn't considered a compile-time constant
        break;

    case null:   // won't compile, there can't be a null case
        break;
}
```

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?

While Loop

Syntax:

```
while (boolexpr)
 stmt
```

Semantics:

- evaluate boolexpr. If true, execute stmt and retry. If false, skip stmt (while loop is done).
- if boolexpr is false on first time, stmt is never run!
- no 'real-time' checking on boolexpr during stmt's evaluation: its value is only checked between iterations.
- if no part of stmt makes boolexpr become false, the loop is infinite.

Example:

```
while (x<100)
    System.out.println(x++);
```

→

```
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; if true, repeat (evaluate stmt again). If false, do-while is done.
- semicolon ; after (boolexpr) is required!
- stmt runs at least once

Example:

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

For Loop

Syntax:

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

Semantics:

- **initializer** is a statement. Runs exactly once, before everything else. (If a variable is declared, its scope is only within loop. Variable doesn't have to be declared, it can already exist).
- **guard** is a boolean expression. Each iteration (including first), this is checked: true => run stmt; false => exit loop.
- **update** is a statement. Runs after the body_stmt
- Note: initializer, guard, and update could each be omitted!
  E.g., for (;;) stmt

Example:

```
for (int i = 0; i<10; i=i+1)
    System.out.println(i);
```

Understanding the For Loop

The following two pieces of code would run **identically**:

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

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

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.

Iterator-based For Loop

- http://docs.oracle.com/javase/1.5.0/docs/guide/language/foreach.html
- Some Java classes behave as "Iterators".
  - **must have these methods: hasNext(), next(), remove()**

Syntax: for (Type identifier : iterExpr)
stmt

Semantics:

- iterExpr must be an iterable type (supplies values of type Type)
- identifier can be used in stmt
- each item in iterExpr is used as ident's value in its own iteration.
- Ordering is based on hasNext/next implementations.

Example:

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

Iterators and Looping

- Again, why so many flavors of iteration?
  - What are the benefits?
  - What are the drawbacks?
  - How would you design your own language?

- Some other control flow statements
  - `break` (immediately leave nearest loop)
  - `continue` (immediately skip to next iteration of loop)
  - `return` (immediately exit a method)