<< Lecture 4 >>

Short announcements

- Change in office hours: **Monday: 1:30-2:30pm**, Wednesday-Thursday: 3:30-4:30pm
- **Discussion this week**
- TA office hours will be announced soon—likely to be in 201N MLH
- **Homework 1 to be announced this week**
- Quizzes likely this and/or next week
- **Midterms projections:**
  - M1: Second week of October (wk 41)
  - M2: Third week of November (wk 47)
  - Out-of-date testing: only with medical or athletic reasons
- **Assignments: 5 in total**

Today's Plan

- Continue on Java's basics, including:
  - ~~basic types (aka primitive types)~~
  - ~~arrays~~
  - control flow
    - conditionals
    - loops
  - classes (as modules)

If statements

- The **if statement** is the most basic control flow structure
- Usage: **if (***guard***)** {true case}
  - The **guard** is or generates a boolean value
- Or: **if (**guard**)** {true case} **else** {false case}
- **The parentheses are mandatory**
- Brackets unnecessary **if** block = one instruction

- If statements can consume several lines

```
static void oddEven(int x) {
  if (x%2==0)
      System.out.println("Even");
  else
      { System.out.println("Odd"); }
}

public static void main(String[] x) {
  oddEven(-3);
  oddEven(-2);
  oddEven(-1);
  oddEven(0);
  oddEven(1);
  oddEven(2);
  oddEven(3);
}
```

- But we can try more conditions in succession rather than just one
- The following example translates age to stages of life (roughly classified)

```
static String ageText(int age) {
  String ans = "";
  if (age<0)
      ans = "Unborn--fetus?";
  if (age>=0 && age<12)
      ans = "Childhood";
  if (age>=12 && age<20)
      ans = "Adolescence";
  if (age>=21 && age<65)
      ans = "Adulthood";
  if (age>=65)
      ans = "Mature Adulthood";
  return ans;
}
```

- Observe that we can rewrite the above **if** statements in the following manner—*why?*

```
static String ageText(int age) {
  String ans = "";
  if (age<0)
      ans = "Unborn--fetus?";
  else if (age<12)
      ans = "Childhood";
  else if (age<20)
      ans = "Adolescence";
  else if (age<65)
      ans = "Adulthood";
  else
      ans = "Mature Adulthood";
  return ans;
}
```

- Yet another way to rewrite the above code is:

```
static String ageText(int age) {
  if (age<0)
      return "Unborn--fetus?";
  else if (age<12)
      return "Childhood";
  else if (age<20)
      return "Adolescence";
  else if (age<65)
      return "Adulthood";
  else
      return "Mature Adulthood";
}
```

- Note that the previous method has a type other than **void**; therefore, **it must end in a return statement**—*why?*
  - **The return statement can be inside the last else of the last if statement**, which should be the last instruction in the method, anyway.
  - If you forget, Netbeans will remind you; Netbeans alerts the programmer about issues with the correctness of the code
  - Netbeans can also suggest code, as we will see later; this is possible because Java is strongly, statically typed (unlike Python)

Inline if statement

- **If**s can *generate* values now, but they always have an else statement—*why?*
- Syntax: guard**?true_value:else_value**

```
String agetxt = (age<21)?"Young":"Old";
String oddeven = (num%2==0)?"Odd":"Even";
```

- Inline **if** statements are useful for writing smaller code, but they are not necessary

Switch statement

- Switch statements are a great space saver for when testing strict equalities (==)

```
switch( value ) {
  case X:  // occurs if value==X
    …      // here you put your code
    break; // immediately exists the switch
  case Y:
    …
    break;
  …
  default: // behaves like last else
    …      // no break here?-Why?
}
```

Problems about if-Switch statements

1. Is it possible to *reduce* all **if** and **switch** statements to the simple **if** statement without else? If so, show how; if not, argue why.
2. Is it possible to *reduce* all **if** statements to **switch** statements? If so, show how; if not, argue why.
3. Write a static method that reproduces the behavior of the inline **if** statement. The method **should not** declare variables inside.

While loops

- The basic structure to repeat instructions is the **while** loop, but the most used is the **for** loop
- Syntax of the **while** loop:
  - **while (**guard**)** {instructions to repeat}
  - The block will be repeated as long as the condition evaluates to **true**
  - The **condition** or **guard** is evaluated in between executions of the blocks

```
int i = 0;
while ( i < 10 ) {
  System.out.println("In this block, i = "+i);
  i = i + 1;
}
```

- To make a while loop run forever, we just write **while(true)**... Fortunately, Netbeans has a stop button
- Note that in the above example, we initialize a variable (i), we then use it in the guard (i < 10), and then we update it (i = i + 1)

For loops

- The above *use case* is pretty typical, which is why we have the **for** loop
- Syntax of the **for** loop:
  - **for(** init_line **;** guard **;** update_line **)** {instructions to repeat}

```
for(int i=0; i<10; i=i+1)
  System.out.println("In this block, i = "+i);
```

- By the way, we can replace **i=i+1** by **i**++ or ++**i**
- **for** and **while** loops are equivalent; in fact, the equivalence is as follows

```
// for loop
for(init_line; guard; upd_line) { some_code; }

// while loop
init_line;    //note that the init_line is only 1 instruction!
while(guard) {
   some_code;
   upd_line;  //note that the upd_line is only 1 instruction!
}
```

Control during while and for loops

- One can also exit the loop instantly
- **break** allows to instantly end the block and stop the execution of the loop
  - We saw this one in the switch statement
- **continue**, on the other hand, instantly ends the block and returns to the guard, to see if the loop should continue
- *These two can be used in **many** exercises*

For loops and arrays

- **for** loops go well with arrays:

```
char[] vowels = {'a','e','i','o','u'};

for(int i=0; i<vowels.length; i++)  System.out.println( vowels[i] );
```

- We can also use the simpler version:

```
char[] vowels = {'a','e','i','o','u'};

for(char c: vowels)  System.out.println( c );
```

Accessing "modules"

- In Java modules are either:
  - Classes
  - Packages, which contain classes or other packages
- **Let us create a simplified input-output class**
- Let us make it **static**
  - This will be a rare choice later on

```
...
```

<< Lecture 5 will resume from here >>