Outline

- The *if* statement
- Comparing numbers and strings
- Nested Branches
- Boolean Variables
- The *while* and *for* Loops
  - Section 3.1–3.4, 3.7, 4.1, 4.3
The if Statement

The if statement allows a program to carry out different actions depending on the nature of the data to be processes

```c
int actualFloor;
if (floor > 13) {
    actualFloor = floor - 1;
} else {
    actualFloor = floor;
}
```

conditional operator:
```c
int actualFloor = (floor > 13) ? floor - 1 : floor;
```
The if Statement

The else branch is optional

```java
int discountedPrice = originalPrice;
if (originalPrice > 100) {
    discountedPrice = originalPrice - 10;
}
```

**NB:** do not put the semicolon after `if(...)`

```java
int discountedPrice = originalPrice;
if (originalPrice > 100); { // ERROR
    discountedPrice = originalPrice - 10;
}
```
Comparing Numbers

relational operators  >, >=, <, <=, ==, !=

NB: why not, e.g., => instead of >=? (assignment operators, see p. 16)

use small $\varepsilon = 10^{-14}$ to compare floating numbers:

```java
final double EPSILON = 1e-14; // constant (see p. 5)
double r = Math.sqrt(2.0);
// r*r is 2.0000000000000004
if (Math.abs(r * r - 2.0) < EPSILON) {
    System.out.println("Math.sqrt(2.0) squared is approx 2.0");
}
```

relational operators have lower precedence than arithmetical operators, e.g., boolean v = floor - 1 < 13;
Constants

- the reserved word `final` ensures that the value of the variable never changes:

  ```java
  final double BOTTLE_VOLUME = 2;
  ```

- use names for constants and avoid “magic numbers”

- “all uppercase with words separated by _” is a **coding convention**

Java does not check it!
IEEE Floating Point Numbers

- **float**: $\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$ with approx. 7 decimal digits

<table>
<thead>
<tr>
<th>sign</th>
<th>exponent (8 bit)</th>
<th>fraction (23 bit)</th>
</tr>
</thead>
</table>

**NB:** float should never be used for precise values, e.g., currency; use java.math.BigDecimal class instead

- **double**: $\pm 2.23 \times 10^{-308}$ to $\pm 1.80 \times 10^{308}$ with approx. 15 decimal digits

<table>
<thead>
<tr>
<th>sign</th>
<th>exponent (11 bit)</th>
<th>fraction (52 bit)</th>
</tr>
</thead>
</table>
Strings

- strings are sequences of characters:
  ```java
  String name = "Harry";
  ```

- the length method yields the number of characters in the string:
  ```java
  int n = name.length();
  ```

- the empty string "" is of length 0

- use the + operator to concatenate strings
  ```java
  String lastname = "Morgan";
  String fullname = name + " " + lastname;
  ```

**NB:** whenever one of the arguments of + is a string, the other argument is converted to a string
Substrings

- string that is made up of the characters starting at position $i$ and containing all the characters up to, but not including, the position $j$:  
  
  ```java
  String greeting = "Hello!";
  String sub = greeting.substring(0,2);
  ```

- all characters from the position $i$ to the end of the string:  
  
  ```java
  String tail = greeting.substring(2);
  ```

- these are all examples of **instance methods** of the class `String`
Comparing Strings

"Tomato".substring(0,3).equals("Tom") is true

NB: == not useful for strings:
"Tomato".substring(0,3) == ("Tom") is false

why?

Strings are objects

s1.compareTo(s2) compares strings lexicographically:
< 0 if s1 comes before s2 in the dictionary
== 0 if s1 equals s2
> 0 if s1 comes after s2 in the dictionary
Strings and Characters

- **Strings and Characters**

  string **positions** are counted starting with 0

  ```java
  char start = name.charAt(0);          // 'H'
  char last = name.charAt(name.length() - 1); // 'y'
  ```

  - do not confuse characters ('H')
    and strings containing a single character ("H")
The class `Character` provides the following methods:

- `Character.isDigit(ch): '0', '1', ..., '9', ...`
- `Character.isLetter(ch): 'A', 'B', ..., 'Z',
  'a', 'b', ..., 'z', ...`
- `Character.isUpperCase(ch): 'A', 'B', ..., 'Z', ...`
- `Character.isLowerCase(ch): 'a', 'b', ..., 'z', ...`
- `Character.isWhiteSpace(ch): ' ', '
', ...`
What is wrong in the following example?

```java
double shippingCharge = 5.00;
if (country.equals("USA"))
  if (state.equals("HI"))
    shippingCharge = 10.00; // Hawaii is
    // more expensive
  else
    shippingCharge = 20.00; // as are foreign
    // shipments

NB: use curly brackets {} to avoid
the dangling else problem
```
The Boolean type `boolean` has two values, `false` and `true`.

Three Boolean operators that combine conditions:

- **`&&`** (and)
- **`||`** (or)
- **`!`** (not)

### Truth Table

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>A &amp;&amp; B</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

| A   | B   | A || B |
|-----|-----|-------|
| false | false | false |
| false | true | true  |
| true  | false | true  |
| true  | true | true  |

<table>
<thead>
<tr>
<th>A</th>
<th>!A</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>
Boolean variables and Operators

&& and || are computed using **lazy evaluation**: stops as soon as the result is known

`quantity > 0 && price/quantity < 10` never divides by 0

De Morgan’s Laws:  

! (A && B) is equivalent to  !A || !B  

! (A || B) is equivalent to  !A && !B

**NB:** Java does not use mathematical notation:

```
if (0 <= temp <= 100) // ERROR
```
The while Loop

a loop executes instructions repeatedly while a condition is true

```java
while (balance < TARGET) {
    year++;
    double interest = balance * RATE / 100;
    balance += interest;
}
```

<table>
<thead>
<tr>
<th>year</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1000.00</td>
</tr>
<tr>
<td>1</td>
<td>1030.00</td>
</tr>
<tr>
<td>2</td>
<td>1060.90</td>
</tr>
<tr>
<td>3</td>
<td>1092.73</td>
</tr>
<tr>
<td>4</td>
<td>1125.51</td>
</tr>
</tbody>
</table>
Assignment Operations

- shortcuts for increment and decrement:
  
  ```
  i++; is the same as i = i + 1;
  i--; is the same as i = i - 1;
  ```

- mixing operations and assignment:

  ```
  i += 2; is the same as i = i + 2;
  i *= 2.5; is the same as i = i * 2.5;
  ... 
  ```

- `+=, etc. are of lowest precedence:

  ```
  i /= 2 + 3; is the same as i = i / (2 + 3);
  ```
Scope of a Variable

- The scope of a variable is the part of the program in which it is **visible**
  - the entire method of a method’s parameter variable
  - from its declaration until the end of the block, for a local variable
  - the **for** statement, for a local variable declared in a **for** statement

- Two variables can have the same name provided their scopes **do not overlap**
What is wrong here?

```java
public static int sumOfSquares(int n) {
    int sum = 0;
    for (int i = 1; i <= n; i++) {
        int n = i * i;
        sum = sum + n;
    }
    return sum;
}
```
Loops and Assignments

```java
int i = 6;
while (i >= 0) {
    System.out.println(i - 1);
    i = i - 2;
}
```

<table>
<thead>
<tr>
<th>i</th>
<th>i &gt;= 0</th>
<th>i - 1</th>
<th>i - 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>true</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>true</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>true</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>true</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>-2</td>
<td>false</td>
<td>END OF THE LOOP</td>
<td></td>
</tr>
</tbody>
</table>

SP1 2015-03
The for Loop

1. **for** (int k = 2; k <= 9; k++) {
2.     String s = s0;
3.     if (k % 2 == 1)
4.         s = s1;
5.     System.out.println(k + " is " + s);
6. }

**initialisation (statement)**

**condition boolean expression**

**update (statement)**
```java
int k = 2;
while (k <= 9) {
    String s = s0;
    if (k % 2 == 1)
        s = s1;
    System.out.println(k + " is " + s);
    k++;
}
```
Collatz conjecture

The sequence $a_{n+1} = \begin{cases} a_n/2, & \text{if } a_n \text{ is even} \\ 3a_n + 1, & \text{if } a_n \text{ is odd} \end{cases}$

eventually reaches 1 regardless of which positive integer $a_0$ is chosen.

1 while (a > 1) {
2     if (a % 2 == 1)
3         a = 3 * a + 1;
4     else
5         a = a / 2;
6 }
Take Home Messages

- The *if* statement allows a program to carry out different actions depending on the data to be processed.
- Relational operators are used to compare numbers.
- Use *equals* and *compareTo* to compare strings.
- *else* is matched with the preceding *if*.
- *&&* and *||* are computed lazily.
- The *while* loop executes instructions repeatedly while a condition is true.
- The *for* is used when a value runs from a starting point to an ending point with a constant increment.