Java Constructs

Operators (relational, logical)

Conditionals

Loops via iteration
Java has...

- Variable declaration
  - int x;

- Assignment statements (“gets”)
  - x = 5;

- Conditional statements
  - if (x == 5) { x = x+1;} else { x = x-1;} //We don’t like five!

- Loops
  - while (x >= 5) { x = x-1; }

- Functions (aka: Methods)
  - public static int increment(int x) { return x+1; }
Java has a boolean type that can take the value true or false

Booleans arise naturally when using relational operators to compare two values

3 < 5
3 < 2
3 > 2
5 <= 1
5 >= 1
5 == 5
5 == 6
5 != 6
Logical Operators

- Boolean values can be manipulated with the logical operators ! (not), && (and), and || (or)

\[
\begin{align*}
! (3 < 5) & \quad \text{false} \\
! (3 == 5) & \quad \text{true} \\
(3 > 5) \land (7 < 8) & \quad \text{false} \\
(3 < 5) \land (7 < 8) & \quad \text{true} \\
(3 > 5) \lor (7 < 8) & \quad \text{true} \\
(3 > 5) \lor (7 > 8) & \quad \text{false}
\end{align*}
\]
Predicates

- A **predicate** is any method that returns a **boolean** value

  //determine if n is even
  public static boolean isEven(int n) {
    return (n % 2) == 0;
  }

  //determine if num is divisible by factor
  public static boolean isDivisibleBy(int num, int factor) {
  }

  //determine if n is between lo and hi
  public static boolean isBetween(double n, double lo, double hi) {
  }
Predicates

Write your own predicate to determine if n is odd. Then, can you write it another way?

```java
public static boolean isOdd(int n) {
    return (n % 2) == 1;
}
```

```java
public static boolean isOdd(int n) {
    return !isEven(n);
}
```
To choose between two courses of action, to control the program flow, we use *conditional statements* such as `if`, `else if`, and `else`

```java
//returns absolute value of n
public static double abs(double n)
{
    if (n < 0) {
        return -n;
    } else {
        return n;
    }
}

//returns absolute value of n
public static double abs(double n)
{
    // Code snippet
}
```
1. public static void main(String[] args) {
2.     int x = 28; String s = "meow";
3.     if(x < 30 && s.length() < 10) {
4.         x = x + 5;
5.         int y = s.length();
6.         if(x+y > 36) {
7.             System.out.println("hello " + x);
8.         } else if(x+y < 33) {
9.             System.out.println("howdy " + y);
10.        } else {
11.            System.out.println("hi!");
12.        }
13.    } else {
14.        x = x - 10;
15.        int y = s.length() + 5;
16.        if( x == 15) System.out.println("Salut " + x);
17.        else System.out.println("Ciao " + y);
18.    }
19. }
Iteration refers to a sequence of steps that is repeated until some stopping condition is reached.

**while** (boolean_expression)

(1) evaluate boolean expression

(2) if true, execute body of loop and go back to step (1)

statement 1;

statement 2;

...

(3) if false, go to statement after the while

```java
int i = 1;
while (i < 4) {
    System.out.println("CS230");
    i = i + 1;
}
```
Iteration – **for** loop

Execute this statement once before entering loop

```
for (initialization of index variable; boolean expn; update index variable) {
    statements
}
```

If true, execute body of loop

If the boolean expression was true, the body of the loop will be executed

If the boolean expression evaluates to false, drop down to here

```
for (int i = 1; i < 4; i++) {
    System.out.println("CS230");
}
```

Write a for loop that prints the numbers from 1 to 10.

```
for (int i = 1; i <= 10; i++) {
    System.out.println(i);
}
```
Do-loop vs While-loop vs For loop

1. Do-loop:
   - **Initialization**:
   - **Condition Evaluated**:
   - Statement
   - If true, go back to condition evaluated; if false, do not execute statement.

2. While-loop:
   - **Condition Evaluated**:
   - Statement
   - If true, do not execute condition evaluated; if false, exit.

3. For-loop:
   - **Initialization**:
   - **Condition Evaluated**:
   - Statement
   - Increment
   - If true, go back to condition evaluated; if false, exit.
Data Types in Java
**Data Types in Java**

- Java is a **statically typed** language
  - You must define the type of each variable when it is declared

- Unlike Python, not all variables in Java are objects
  - Some are **primitive data types** (but have related objects)

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Storage</th>
<th>Range of values</th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>32 bits</td>
<td>-2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>long</td>
<td>64 bits</td>
<td>-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807</td>
</tr>
<tr>
<td>float</td>
<td>32 bits</td>
<td>Approx. –3.4E+38 to +3.4E+38 with 7 significant digits</td>
</tr>
<tr>
<td>double</td>
<td>64 bits</td>
<td>Approx. –1.7E+308 to +1.7E308 with 15 significant digits</td>
</tr>
<tr>
<td>char</td>
<td>16 bits</td>
<td>65,535 Unicode characters</td>
</tr>
<tr>
<td>boolean</td>
<td>1 bit</td>
<td>true or false</td>
</tr>
</tbody>
</table>
Decimal Numbers

double num = 5.2;
num = 1.4;
num = num * 2.0;
System.out.println(num);

double fahrenheit = 98.6;
double celsius = (fahrenheit - 32) * 5 / 9;
System.out.println(celsius);
Comparing Float Values

- You should rarely use the equality operator (==) when comparing two floating point values (float or double).

- Two floating point values are equal only if their underlying binary representations match exactly.

- Computations often result in slight differences that may be irrelevant.

- In many situations, you might consider two floating point numbers to be “close enough” even if they aren't exactly equal.
Using Objects and Strings
Creating Objects

- Generally, we use the `new` operator to create an object:

  ```java
title = new String("Hello CS230!");
  ```

  This calls the String `constructor`, which is a special method that sets up the object.

- Creating an object is called `instantiation`.
- An object is an `instance` of a particular class.
Creating Strings

- Because strings are so common, we don't have to use the `new` operator to create a `String` object

  ```java
  title = "Java rocks!";
  ```

- This is special syntax that works only for strings

- Each string literal (enclosed in double quotes) represents a `String` object
The String Class

- Once a `String` object has been created, neither its value nor its length can be changed.

- Thus we say that an object of the `String` class is *immutable*.

- However, several methods of the `String` class return new `String` objects that are modified versions of the original.
The String Class

- It is occasionally helpful to refer to a particular character within a string.
- This can be done by specifying the character's numeric index.
- The indexes begin at zero in each string.
- In the string "Hello", the character 'H' is at index 0 and the 'o' is at index 4.
Some methods of the `String` class:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>String (String str)</code></td>
<td>Constructor: creates a new string object with the same characters as <code>str</code>.</td>
</tr>
<tr>
<td><code>char charAt (int index)</code></td>
<td>Returns the character at the specified index.</td>
</tr>
<tr>
<td><code>int compareTo (String str)</code></td>
<td>Returns an integer indicating if this string is lexically before (a negative return value), equal to (a zero return value), or lexically after (a positive return value), the string <code>str</code>.</td>
</tr>
<tr>
<td><code>String concat (String str)</code></td>
<td>Returns a new string consisting of this string concatenated with <code>str</code>.</td>
</tr>
<tr>
<td><code>boolean equals (String str)</code></td>
<td>Returns true if this string contains the same characters as <code>str</code> (including case) and false otherwise.</td>
</tr>
<tr>
<td><code>boolean equalsIgnoreCase (String str)</code></td>
<td>Returns true if this string contains the same characters as <code>str</code> (without regard to case) and false otherwise.</td>
</tr>
<tr>
<td><code>int length ()</code></td>
<td>Returns the number of characters in this string.</td>
</tr>
<tr>
<td><code>String replace (char oldChar, char newChar)</code></td>
<td>Returns a new string that is identical with this string except that every occurrence of <code>oldChar</code> is replaced by <code>newChar</code>.</td>
</tr>
<tr>
<td><code>String substring (int offset, int endIndex)</code></td>
<td>Returns a new string that is a subset of this string starting at index <code>offset</code> and extending through <code>endIndex-1</code>.</td>
</tr>
<tr>
<td><code>String toLowerCase ()</code></td>
<td>Returns a new string identical to this string except all uppercase letters are converted to their lowercase equivalent.</td>
</tr>
<tr>
<td><code>String toUpperCase ()</code></td>
<td>Returns a new string identical to this string except all lowercase letters are converted to their uppercase equivalent.</td>
</tr>
</tbody>
</table>
String s1 = new String("Grace Hopper");
String s2 = "CU L8R";
String s3 = ":);"

System.out.println(s1.toLowerCase());
System.out.println(s1.length());
System.out.println(s2.length());
System.out.println(s2.equals(s3));
System.out.println(s2.equals("CU L8R"));
System.out.println(s2.charAt(1));
System.out.println(s1.substring(7, 11));
System.out.println(s2.substring(0, 2).toLowerCase());
Strings in Java vs Python

- Strings in Java and Python are quite similar.
  - Like with Python, Java strings are immutable.

- The difference is that
  Java uses method calls where Python uses Operators.

<table>
<thead>
<tr>
<th>Python</th>
<th>Java</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>str[3]</td>
<td>str.charAt(3)</td>
<td>Return character in 3rd position</td>
</tr>
<tr>
<td>str[2:5]</td>
<td>str.substring(2,4)</td>
<td>Return substring from 2nd to 4th</td>
</tr>
<tr>
<td>len(str)</td>
<td>str.length()</td>
<td>Return the length of the string</td>
</tr>
<tr>
<td>str.find('x')</td>
<td>str.indexOf('x')</td>
<td>Find the first occurrence of x</td>
</tr>
<tr>
<td>str.split()</td>
<td>str.split(' \s')</td>
<td>Split the string on whitespace into a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>list/array of strings</td>
</tr>
<tr>
<td>str.split(',')</td>
<td>str.split(',')</td>
<td>Split the string at ',' into a list/array of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>strings</td>
</tr>
<tr>
<td>str + str</td>
<td>str.concat(str)</td>
<td>Concatenate two strings together</td>
</tr>
<tr>
<td>str.strip()</td>
<td>str.trim()</td>
<td>Remove any whitespace at the beginning or end</td>
</tr>
</tbody>
</table>
The Java API
The Java API

- A class library is a collection of classes that we can use when developing programs.

- The Java API is the standard class library that is part of any Java development environment.

- API stands for Application Programming Interface.

- Various classes we've already used (System, Scanner, String) are part of the Java API.

- Other class libraries can be obtained through third party vendors, or you can create them yourself.
Packages

- The classes of the Java API are organized into packages

<table>
<thead>
<tr>
<th>Package</th>
<th>Provides support to</th>
</tr>
</thead>
<tbody>
<tr>
<td>java.applet</td>
<td>Create programs (applets) that are easily transported across the Web.</td>
</tr>
<tr>
<td>java.awt</td>
<td>Draw graphics and create graphical user interfaces; AWT stands for Abstract Windowing Toolkit.</td>
</tr>
<tr>
<td>java.beans</td>
<td>Define software components that can be easily combined into applications.</td>
</tr>
<tr>
<td>java.io</td>
<td>Perform a wide variety of input and output functions.</td>
</tr>
<tr>
<td>java.lang</td>
<td>General support; it is automatically imported into all Java programs.</td>
</tr>
<tr>
<td>java.math</td>
<td>Perform calculations with arbitrarily high precision.</td>
</tr>
<tr>
<td>java.net</td>
<td>Communicate across a network.</td>
</tr>
<tr>
<td>java.rmi</td>
<td>Create programs that can be distributed across multiple computers; RMI stands for Remote Method Invocation.</td>
</tr>
<tr>
<td>java.security</td>
<td>Enforce security restrictions.</td>
</tr>
<tr>
<td>java.sql</td>
<td>Interact with databases; SQL stands for Structured Query Language.</td>
</tr>
<tr>
<td>java.text</td>
<td>Format text for output.</td>
</tr>
<tr>
<td>java.util</td>
<td>General utilities.</td>
</tr>
<tr>
<td>javax.swing</td>
<td>Create graphical user interfaces with components that extend the AWT capabilities.</td>
</tr>
<tr>
<td>javax.xml.parsers</td>
<td>Process XML documents; XML stands for eXtensible Markup Language.</td>
</tr>
</tbody>
</table>
When you want to use a class from a package, you could use its **fully qualified name**

```java
java.util.Scanner
```

Or you can **import** the class, and then use just the class name:

```java
import java.util.Scanner;
```

To import all classes in a particular package, you can use the `*` wildcard character:

```java
import java.util.*;
```
The java.lang Package

- All classes of the java.lang package are imported automatically into all programs
- It's as if all programs contain the following line

  ```java
  import java.lang.*;
  ```
- That's why we didn't have to import the System or String classes explicitly in earlier programs
- The Scanner class, on the other hand, is part of the java.util package, and therefore must be imported
Scanner Class

```java
// GasMileage.java  Java Foundations
// Demonstrates the use of the Scanner class to read numeric data.

import java.util.Scanner;

public class GasMileage {

    public static void main (String[] args) {
        int miles;
        double gallons, mpg;
        Scanner scan = new Scanner (System.in);
        System.out.print ("Enter the number of miles: ");
        miles = scan.nextInt();
        System.out.print ("Enter the gallons of fuel used: ");
        gallons = scan.nextDouble();
        mpg = miles / gallons;
        System.out.println ("Miles Per Gallon: " + mpg);
    }
}
```
Math Class

System.out.println(Math.max(100, 50));
System.out.println(Math.sqrt(25));
System.out.println(Math.log(10));

// Given area of circle, returns the circle's radius.
// Since area=πr², we have r = squareRoot(area/π).
public static double getCircleRadius(double area) {
    return Math.sqrt(area/Math.PI);
}

System.out.println(getCircleRadius(100));