Java Overview

---

1. BASIC TYPES

—`boolean`: { true, false}
—`char`: any character
—`int`: $\{-2^{31}, 2^{31}-1\}$
—`long`: $\{-2^{63}, 2^{63}-1\}$
—`float`:$\{-10^{38}.....10^{38}\}$ or so
—`double`: even bigger

2. DECLARING VARIABLES

```
<type> <variable-name>;
```

—`<type>` is either a basic (primitive) type or a non-primitive type (class).
—Variable names should make sense.
—All class variables must be commented.
—Declaring CONSTANTS:

```
static final int MONDAY = 1;
```

—To refer to a character use single quotes: `char a = 'a';`. Double quotes are used for Strings: `String s = ''s''`.
—`int` and `long`: both hold integers, but longs can hold bigger integers.
—float and doubles both hold reals, but doubles are bigger.

3. OPERATORS (IN ORDER OF PRECEDENCE)

(1) `++`
(2) `--`
(3) `!`
(4) `*, /, %`
(5) `+, -`
(6) `<, <=, >=.  >`
(7) `==, !=`
(8) `&&, ||`
(9) `=`

Note: a/b returns an integer (truncated value of a/b) if both a,b are integers. There are also Math operators:

—`Math.sqrt(x)`
—`Math.abs(x)`
—`Math.pow(x,y)`
—`Math.sin(x)` and other trigonometric functions
—`Math.round(double x)`

4. TYPE CASTING

```
double pi = 3.14;
int i;
i = (int) pi;
```

Sometimes Java will do casts automatically. Better style is to do the cast explicitly yourself:

Even better style is to avoid casts like this one altogether, if you can.

Casting is a powerful feature of Java and we'll come back to it later.

5. ARRAYS

```
int arr1[];
arr1 = new int [10];

int arr2[] = {1, 2, 3, 4};
int arr3 = new int [100];
int arr4[] = arr3;
```

—Don't forget: arrays are 0-based.

—Discuss the implications of the last statement.

—You can ask for an array length: `arr1.length`. Note the lack of parentheses.

—Exercise: initializing an array; copying an array; printing an array.

—Passing an array as parameter:

```
public double largest(double [] elt) {

}

double arr1[];
...
blah.largest(arr1);
```

Note: you are passing the array, NOT a copy of it. If you modify the array inside the method...the changes stay.

—Exercise: a method that finds the largerst element in an array.

6. CONDITIONALS

```
if (condition) {
    <statements>;
};

if (condition) {
    <statements>;
} else {
    <statements>;
}
```

```
switch (expression) {
  case <constant>: <statement>; break;
  default: <statement>; break;
}
```

Example: Craps (see website for demo)

```
if ( newGame ) {                      // start a new game
     if ( roll == 7 || roll == 11 ) {  // 7 or 11 wins on first throw
          status.setText( "You win!" );
      }
      else if ( roll == 2 || roll == 3 || roll == 12 ) {  // 2, 3, or 12 loses
          status.setText( "You lose!" );
      }
      else {                        // Set roll to be the point to be made
          status.setText( "Try for your point!" );
          point = roll;
          newGame = false;          // no longer a new game
      }
}
else {     // continue trying to make the point
      if ( roll == 7 ) {           // 7 loses when trying for point
          status.setText( "You lose!" );
          newGame = true;          // set to start new game
       }
       else if ( roll == point ) { // making the point wins!
          status.setText( "You win!" );
          newGame = true;
       }
       else {                      // keep trying
          status.setText( "Keep trying for " + point + " ..." );
       }
}
```

—Note$_1$: You don't need to do:

```
if (boxGrabbed == true)
```

You can simply say

```
if (boxGrabbed)
```

—Note$_2$: Instead of

```
if (box.contains(point)) {
   boxGrabbed = true;
} else {
   boxGrabbed = false;
}
```

you can simply say

```
boxGrabbed = box.contains(point);
```

7. LOOPS

```
for (<initialize>; <test>; <iteration>) {
    <statements>;
}

while (condition) {
    <statements>;
}
```

Example 1: what prints?

```
for (int i=0; i<3; i++)
   for (int j = 0; j<3; j++)
        System.out.println(i + " " + j);
```

Example 2: what prints?

```
for (int j=0; j<3; j++)
   for (int i = 0; i<3; i++, j++)
        System.out.println(j + " " + i);
```

Example 3: what prints?

```
int j=4;
for (int i=0; i<j; i++,j=j+4)
   for (; j>2; j=j/2)
        System.out.println(j + " " + i);
```

Example 4: what prints?

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

8. CLASSES

A class is a recipe for creating objects. A class (1) stores things (2) has operators.
Typically the data stored is more than just a single variable.
The operators are (called) *methods*.

```
class Bicycle {

       private int speed;
       private int gear;

       public  Bicycle() {
           speed = 0;
           gear = 0;
       }
       public void changeGear(int newValue) {
            gear = newValue;
       }
       public void speedUp(int increment) {
            speed = speed + increment;
       }
       public void applyBrakes(int decrement) {
            speed = speed - decrement;
       }
       public void printStates() {
            System.out.println("speed:"+speed+" gear:"+gear);
       }
}
```

—data
  —private: can be accessed from anywhere within the class definition, but not outside
  —public: can be accessed from anywhere
  —protected: can be accessed inside the class, and in teh subclasses of this class
—methods: can be public, private or protected
  —accessors (getters)
  —mutators (setters)
  —both
—constructor(s)
  —method with same name as the class called to create a new instance of the class

9. METHODS

Every method has an associated class; in other words, methods are defined only within classes (not standalone).

Within the class, a method is invoked by calling its name.
Between classes you must use ".": `<classname>.methodname(<arguments>)`.
Methods are usually `public`. That is, any other class can call the method.

```
public <return-type> <methodname> (<paramater list>) {
...
}
```

Methods that are "internal" to the class and are not to be called from outside should be declared `private`.

```
private <return-type> <methodname> (<paramater list>) {
...
}
```

Methods can be called only on instances of a class (unless declared static as below). That is, you need to first create an instance of class in order to call a method on it.

```
Bicycle b = new Bicycle();
b.changeGear(2);
```

Within the class, a method is invoked by calling its name `<methodname>(<arguments>)`.. Between classes you must use the dot operator ".": `<object>.<methodname>(<arguments>)`.

10. CLASSES VS. OBJECTS

A class is a recipe for creating objects. Objects are instances of classes. An object is created by calling the constructor of a class— this is called *instantianting* a class.

```
// Create two different Bicycle objects
Bicycle bike1 = new Bicycle();
Bicycle bike2 = new Bicycle();

// Invoke methods on those objects
bike1.speedUp(10);
bike1.changeGear(2);
bike1.printStates();

bike2.speedUp(10);
bike2.changeGear(2);
bike2.speedUp(10);
bike2.changeGear(3);
bike2.printStates();
```

11. CONSTRUCTORS, MULTIPLE CONSTRUCTORS

A constructor is a method that is invoked to create an object, like this:

```
SomeClass x = new SomeClass();
```

The constructor has no return type. It is implicitly `void`.

A class may have no constructor. In that case, Java provides a default "dummy" constructor (i.e. does nothing).

A class may have multiple constructors. When the user creates an object, the right constructor will be invoked that matches the types of the arguments. However, there cannot be two constructors with the same *signature* (list of parameter types).

```
class Person {
   private String firstName, lastName;
   private int id;
   private String address;

   public Person(String fname, String lname)
   public Person(int pid)
   public Person(String address)
}

Person a, b, c;
a = new Person("Howard", "Zinn");
b = new Person(47561845);
c = new Person("8650 College Street Brunswick 04011 USA");
```

12. MAIN METHOD

Some classes are meant to be utilized by other classes. Others are meant to be stand-alone programs. A class that is a standalone program, i.e. the user can "run" it, it must have a `main` method:

```
public class AClass {
   public static void main(String args[]) {
       //this is the method that gets executed when the user runs the class
       //the body of the method
       ...
   }
}
```

Suppose we first compile this class:

```
javac AClass.java
```

Then we can run it:

```
java AClass
```

In Java, when you execute a class, the system starts by calling the class `main` method. Note that main is static, that is it does not need an instance of the class.

13. HELLO WORLD

To create objects in Java you first have to define a class. A class is an encapsulation of data and methods: For example:

```
//import java.io.*
//import java.util.*

public class  HelloWorld {

  //data
  //methods
```

```
  public static void main(String[] args) {
     System.out.println("Hello world!");
  }
}
```

—Java packages: `java.lang` (automatically imported; contains e.g. `System` and `String`). `java.util` contains for e.g. random nb generator.

—when the application is run, `main` is executed.

—if the name of your class is Xxx, then it should occur in a file called Xxx.Java

—comments: // or /* .... */

—public

14. COMMON CLASSES: CLASS STRING

Defined in Java package `java.lang` which is included by default in any of your programs.

```
String s = "dog";
s = "hot " + s;
System.out.println("the length of " + s + " is " + s.length());

String t;
t = new String("this is an example");

String u;
//before u is initialized, u is null
//calling any method on a null string is illegal and will result in a run-time error
```

Operators (methods):

—`s.length()`

—`s.charAt(i)`

—`s.compareTo(String t)`

—`s.equals(String t)`

—`s.indexOf(String t)`

—`s.substring(start, end)`

You can also put `ints` and `doubles` into `String` expressions:

```
System.out.println("The x coordinate is " + x + "; the y coordinate is " + y);
```

15. PROGRAMMING STYLE

Naming conventions: classes start with Caps; variables start with small, constants ALL CAPS, methods start with small.

For more guidelines see handouts.