Java & Object-Oriented Programming

Java is a fairly new and very popular programming language designed to support secure, platform-independent programming.

It is a good alternative to C or C++, trading a bit of efficiency for easier programming, debugging and maintenance.

Java is routinely interpreted (at the byte-code level), making it significantly slower than compiled C or C++. However, true Java compilers exist, and are becoming more widespread. (IBM's Jalapeno project is a good example). When compiled, Java's execution speed is close to that of C or C++.

Basic Notions

In Java data is either primitive or an object (an instance of some class).

All code is written inside classes, so Java programming consists of writing classes.

Primitive data types are quite close to those of C or C++:

- boolean (not a numeric type)
- char (Unicode, 16 bits)
- byte
- short
- int
- long (64 bits)
- float
- double
Objects

- All Java objects are instances of classes.
- All objects are heap-allocated, with automatic garbage collection.
- A reference to an object, in a variable, parameter or another object, is actually a pointer to some object allocated within the heap.
- No explicit pointer manipulation operations (like * or -> or ++) are needed or allowed.
- Example:

```java
class Point {
    int x, y;
}
Point data = new Point();
```

- Declaring an object reference (like class Point) does not automatically allocate space for an object. The reference is initialized to null unless an explicit initializer is included.
- Fields are accessed just as they are in C:

```java
data.x references field x in object data.
```
- Object references are automatically checked for validity (null or non-null). Hence

```java
data.x = 0;
```
forces a run-time exception if data contains null rather than a valid object reference.

- Java makes it impossible for an object reference to access an illegal address. A reference is either null or a pointer to a valid, type-correct object in the heap. (This makes Java programs far more secure and reliable than C or C++ programs).

Class Members

Classes contain members. Class members are either fields (data) or methods (functions).

Example:

```java
class Point {
    int x, y;
    void clear() {x=0; y=0;}
}
Point d = new Point();
d.clear();
```

A special method is a constructor.
A constructor has no result type. It is used only to define the initialization of an object after the object has been created.
Constructors may be overloaded.

```cpp
class Point {
   int x, y;
   Point() {x=0; y=0;}
   Point(int xin, int yin) {
      x = xin; y = yin;
   }
}
```

Static Members

Class members may be static.

A static member is allocated only once—for all instances of the class.

Ordinary members (called instance members) apply only to a particular class instance (i.e., only one object created from the class definition).

```cpp
class Point {
   int x, y;
   static int howMany = 0;
   Point() {x=0; y=0;
      howMany++;
   }
   static void reset() {
      howMany = 0;
   }
}
```

Static member functions (methods) may not access non-static data. (Why?)

Static members are accessed using a class name rather than the name of an object reference.

For example,

```cpp
Point.reset();
```