Classes and Objects in Java

SYNTAX:

```java
Class <name> {
    <field declarations>
    <method definitions>
}
```

```java
Class Body {
    public long idNum;
    public String nameFor;
    public Body orbits;
    public static long nextID = 0;

    public long getID( ) {
        return idNum;
    }

    public String getName( ) {
        return nameFor;
    }
}
```

Instance Creation with New

```java
Body sun = new Body( );
sun.idNum = Body.nextID++;
sun.NameFor = "Sol ";
sun.orbits = null;

Body earth = new Body( );
earth.idNum = Body.nextID++;
earth.nameFor = "Earth";
earth.orbits = sun;
```

```
Earth
Sol
```
Access to Fields and Methods

- public: accessible anywhere the class is and inherited by subclasses
- private: accessible ONLY in the class itself
- protected: accessible to subclasses and code in the same package and inherited by subclasses
- package: accessible only to code and inherited only by subclasses in the same package

How does this compare to C++?

Defining Constructors

```java
Class Body {
    public long idNum;
    public String name = "unnamed";
    public Body orbits = null;

    private static long nextID = 0;

    Body() {
        idNum = nextID++;
    }

    Body(String bodyName,
         Body orbitsAround) {
        this();
        name = bodyName;
        orbits = orbitsAround;
    }
}
```

*NOTE*: a constructor can invoke another constructor from the same class using `this`. 
J.20

Use of Constructors

constructor with 2 arguments
Body sun = new Body("Sol", null);
Body earth = new Body("Earth", sun);

constructor with no arguments
Body mars = new Body( );
mars.name = "Mars";
mars.orbits = sun;

Overloading: 2 methods with the same name, but different signatures (different number or type of parameters).

When are zero-argument constructors useful?

J.21

The Method toString()

Public String toString() {
    String desc = idNum + " (" + name + ")";
    if (orbits != null)
        desc += " orbits " + orbits.toString();

    NOTE: the toString method is special.

    If you provide a toString( ) method for an object, then it will be used whenever the object is used in a string concatenation.

    System.out.println("Body " + earth);

    What is the output ?
J.22

PARAMETER PASSAGE

In Java, parameters are passed by value.

- Variables containing primitive types cannot be changed by a method.

```java
Class PassByValue{
    public static void main(String[] args) {
        double one = 1.0;
        System.out.println("before: one = "+ one);
        halveIt(one);
        System.out.println("after: one = "+ one);
    }
    public static void halveIt(double arg) {
        arg /= 2.0;
        System.out.println("halved: arg = "+ arg);
    }
}
```

What will the output be?

J.23

- If a variable contains an object reference, the fields of that object can be changed.

```java
Class PassRefByValue {
    public static void main(String[] args) {
        Body sirius = new Body("Sirius", null);
        System.out.println("before: "+ sirius);
        commonName(sirius);
        System.out.println("after: "+ sirius);
    }
    public static void commonName(Body bodyRef) {
        bodyRef.name = "Dog Star";
        bodyRef = null;
    }
}
```

What does this do?

Does the name field of sirius change?

Does the value of sirius change to null?
A member is a field or a method.

A static member is a member that belongs to the class, not to instances of the class.

A static field is just a class variable, such as nextID in class Body. It is assigned its initial value before any instances of the class are created.

A static initialization block can be used to initialize static structures.

   static { <initialization statements> }   

A static method (also called a class method) can be used to modify static fields.

- Java has new, but it doesn’t have free.

- The garbage collector is a system method that finds objects that are no longer referenced and reclaims their memory.

- Garbage collection is especially useful in applications that require linked structures, such as linked lists.

```java
myList = null;
```

Dangling reference

```
null      1    2    27 null
```
EXAMPLE: Using Java to Implement Linked Lists

```java
public class ListNode {
    int Element;
    ListNode Next;
    ListNode(int NewElement, ListNode Node) {
        Element = NewElement;
        Next = Node;
    }
}

public class IntList {
    ListNode Head;
    IntList() {
        Head = new ListNode(0, null);
    }
    void InsertEnd(int NewElement) {
        ListNode Marker;
        for (Marker = Head; Marker.Next != null;
            Marker = Marker.Next);
        Marker.Next = new ListNode(NewElement, null);
    }
    void Delete(int DelElement) throws ListException {
        ListNode Marker;
        for (Marker = Head; Marker.Next != null &&
            Marker.Next.Element != DelElement;
            Marker = Marker.Next);
        if (Marker.Next != null &&
            Marker.Next.Element == DelElement)
            Marker.Next = Marker.Next.Next;
        else
            throw new ListException("Cannot delete:
                element not in list.");
    }
}
```

What does this do when `Delete(17)` is invoked for the list `myList`?

What about when `Delete(1)` is invoked for the list `myList`?

What’s wrong with `myList`?