

Topics: the `for` loop, `String` objects, intro to OOP (object-oriented programming)

Reading (JV): Sec 3.8, 3.9, 2.5, 4.1

The `for` loop

```
for ( initialization; condition; increment )  
    statement;
```

Pattern for doing something n times

```
for ( i=0; i<n; i++ ) {  
    // do something  
}
```

Example 2: Count down

Write a program segment to print “count-down messages.” User enters the number of seconds to go (a positive integer). E.g., if user enters **3**, display the messages

```
T-3 seconds  
T-2 seconds  
T-1 second  
Take-off!
```

Use the **`for`** loop.

```
int t = Keyboard.readInt(); // time left
```

```
System.out.println("T-1 second");  
System.out.println("Take-off!");
```

String (read LL Sec 2.5)

A **`String`** is an *object* (not a primitive data type). You can think of it as a fancy array of **`chars`**.

```
String s = "Hello";           // create a String  
String s2 = s + " class";     // concatenate two Strings  
int x = 100;  
s2 = s2 + x;  
System.out.println(s2);
```

Write another version of the “count down” example using **String** objects explicitly.

```
int t = Keyboard.readInt(); // time left

String mes;                // message to be printed

for ( ; t>0; t-- ) {
    mes = "T-" + t + " second";
    if ( t!=1 )
        mes = mes + "s";
    System.out.println(mes);
}

System.out.println("Take-off!");
```

Object vs Class—Introduction to Object-Oriented Programming (OOP)

Object: contains variables (*fields*, *instance variables*) and methods

- Variables: “state” or “characteristics”, e.g., name, age
- Methods: “behavior” or “action”, e.g., yell, bounce

Class: blueprint (definition) of an object

- No memory space is reserved for object data

Imagine a class `Cookie`. To make a lot of cookies, you may want to

- Make a cookie cutter—define the class
- Stamp out the cookie—instantiate an object

Note that making a cookie cutter doesn’t mean that you have cookies—you *must go through the step of object instantiation after defining the class in order to create actual objects*.

Variables

Two main types of variables:

- Primitive type
- Reference to object

Some variables with different properties:

- **Local variable:** live and die inside a method
- **Instance variable:** owned by and accessed through individual instances (objects)
- **Static variable:** class variable shared by all instances—only one copy in a class