The code here shows a simple class with a constructor and one method.

```java
class Test {
    int i;
    Test(int j) {
        i = j;
    }
    int get() {
        return i;
    }
}
```

A constructor is called when an instance of this class is first created. Here it is used to initialize a variable.

The above code for the constructor,

```java
Test(int j) {
    i = j;
}
```

shows that in the process of creating an instance of the class, an initial value for the instance variable is passed as an argument in the constructor.

Java does not actually require an explicit constructor in the class description. If you do not include a constructor, the Java compiler will create a default constructor in the byte code with an empty argument. This default constructor is equivalent to the explicit `Test()`{}

```java
class Test {
    int i;

    int get() {
        return i;
    }
}
```

A valid class with no constructor explicitly defined.

No constructor is defined so the JVM will use the default constructor:

```java
Test{}()
```

In the discussion of data fields, we noted that the data can receive explicit initial values or default values. You might wonder when does this initialization actually occur?
The java compiler, in fact, puts the initialization of the data into the byte code for the constructor. So, for instance, the above default constructor is equivalent to the one shown below where a constructor explicitly sets the the int variable to 0.

```
class Test {
    int i;
    Test() {
        i = 0;
    }

    int get() {
        return i;
    }
}/
```

This constructor illustrates explicitly the initialization of property values to their default values as would occur if we had just used

```
    Test();
```

or if no constructor had been defined.

Similarly, for a class with several types of data, the initialization of the default or explicit values occurs in the constructor: