Construction time

Declaring and using constructors

An organized way to construct new objects

We have seen that we can create a new object and assign a reference variable like this: `Member myMember = new Member();`

But, what is going on there?

The `new` operator actually calls something called a constructor. A constructor is a block of code in a class with the instructions for creating new instances from the class.

We can write our own constructors for our classes as we'll see soon.

If we don't write any constructors, the compiler will create one for us (but it doesn't do much).

Passing information

In the lecture about instance variable, we had a simple class Member with only one variable, name, and we ended the lecture saying that name should be private.

In order to create a Member instance and give the name variable some value, we would typically use a constructor which can receive the value and use it.

The constructor would typically accept a String reference as a “parameter” and use it to initialize the name variable.

Constructor accepting a String with a name

```
public class Member{
  private String name;

  public Member(String name){ // parameter name
    this.name = name; // save the name!
  }
}

// Code in the other class’ main method:
Member myMember = new Member("Ada"); // provide the name
```

The keyword `this`

The parameter of the constructor was a variable called name. The instance variable was also called name. How do we tell them appart?

The keyword this is used to say:

“This object which is being referred to”

In the case of a constructor, you can think of this as meaning:

“This object which is being constructed here”

The dot after this is used as a separator between the this-reference and some member (which can be private!)

Accepting name, setting this.name

The constructor accepts a reference to a String object and calls the reference "name". In the body of the constructor, this.name (the instance variable name) is set to the same as the parameter name:

```
public Member(String name){
  this.name   =   name;
}
```

instance variable   parameter

The parts of the constructor

First we have an access modifier, in our example “public”. Next we have the class name. Next we have a parameter list (which can be empty but in our example it has one parameter, a String reference called name). Then there’s the body of the constructor. The body is a block of code between { and } .

In the body, we have the code for the initialization, typically using the parameters and saving them in some private instance variables.

```
public Member(String name){
  this.name = name;
}
```

Overloading - more than one constructor

We can provide flexibility to the users of our class by providing more than one way to create a class. We can achieve this by writing more than one constructor. The constructors have to differ and the way then can differ is to declare different parameter lists.

Let’s say our Member class has two instance variables, name and email. We can decide that the user of this class must provide at least a name, but email is optional.

We’d make one constructor accepting only one String as argument, and an additional constructor accepting two Strings (name and email).

Two constructors

```
public class Member{
  private String name;
  private String email;
  public Member(String name){
    this.name=name;
  }

  public Member(String name, String email){
    this(name); // call the other constructor
    this.email = email;
  }
}
```

One constructor calling another

Programmers hate to repeat themselves, so instead of having the same code in both constructors, the second constructor can call the first:

```
public Member(String name){
  this.name=name;
}

public Member(String name, String email){
  this(name); // call the other constructor
  this.email = email;
}
```

Using the two constructors from a test class

```
Member onlyName = new Member("Ada");
Member nameAndEmail = new Member("Eva", "eva@email.com");
```

Remember that variables which are members declared in a class get default values. The first object, referred to by onlyName will have an email variable of value `null` because the constructor which only accepts one String doesn’t set the email variable, only the name. So the email variable is initialized to the default value of all reference variables, which is `null` .

Remember that null was Java’s way of expressing “doesn’t refer to any object”.

String variables are reference variables, so uninitialized String instance variables will have a value of null .

OK, setting variables works, but accessing them?

Now we know that private instance variables can be set using the constructor parameters. But we don’t know how to make use of those variables using a reference to an instance. We can’t do e.g. myMember.name any more, since name is private.

In the next lecture, we’ll see how we can use methods for accessing the state (the values of the variables) of an instance.