Methods

Declaring and calling methods

Objects have a state and a behavior

We have said earlier that objects have a state (they have values stored in instance variables) and a behavior (they can do stuff if we tell them to).

Behavior is expressed in blocks of code called methods.

One use of a method is to get information about the object's state. If we have our old Member example, and we have created a member with a name and an email address (using the constructor which accepts two String references), we still had problem using the value of the object's email for instance.

That was because email was declared private. We could have a method which helps us here. An instance method can use a private variable and even return it to the caller.

Let's look at the code for a method accessing email

```
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;
  }

  public String emailAddress(){
    return this.email; // or shorter: return email;
  }
}
```

Parts of a method

An instance method can have (for now) the following parts:

<access modifier> <return-type> <name>(<parameter-list>){

  <code that does some work>

  <return-statement>

}

If the return-type is void, the method doesn’t give any information back to the caller, it just performs some work.

Access modifier

As with variables, we'll focus only on two access levels, private and public.

An instance method which has access level private, can only be called from code in the same class (the constructor or another instance method).

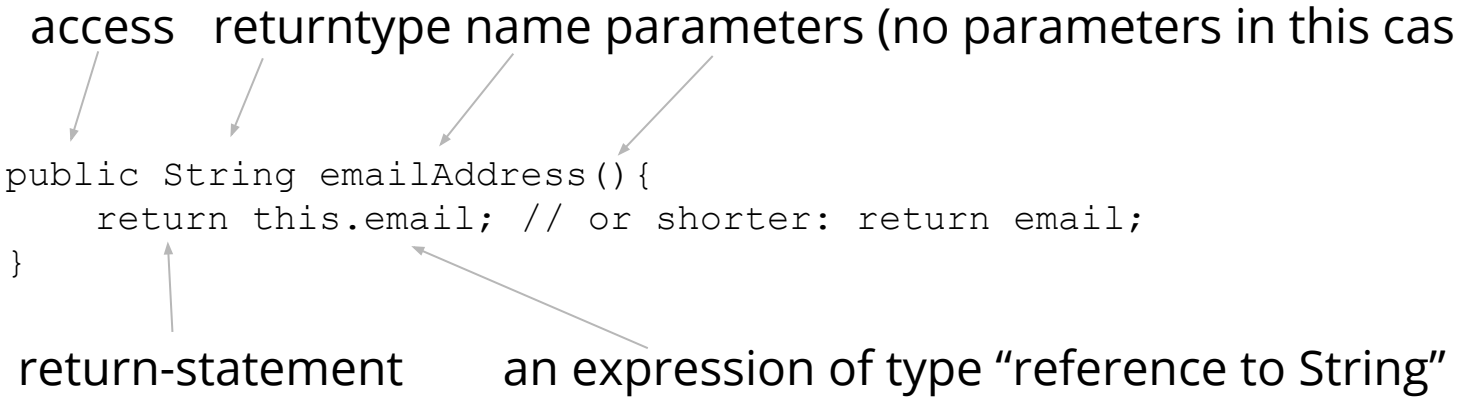
An instance method with public access level can be called from any class (given that a reference to an object is present).

As with public instance variables, a public instance method is called via a reference variable and the dot:

```
System.out.println( myMember.emailAddress() );
```

Return type

Instance methods whose purpose is to give the caller some value back, have a statement using the `return` keyword. The method returns a value, and as with all Java values, they have a type. The method for accessing the email of a member was:



The void return type

The void keyword is actually not really a type. It rather means “there is no return-type of this method, since it doesn’t return any value”.

A method declared “void” only does something, but never returns any value.

You have used a void method many times already! The println() method is declared void. It never returns any value, it just does something, namely prints something.

You access the println method for printing to std out using an object reference called `out`. That reference is declared in the class `System`.

Changing the state of an object

If we decided to allow it, (and that's a big IF), we could provide a method to our Member class allowing users of the class to change the value of the email of a Member object. It's job would be to only do something, not give us a value back, so the method would be declared void:

```
public void changeEmailAddress(String email){
    this.email = email;
}
```

The user of this class would send along a reference to a String with the new email address:

```
myMember.changeEmailAddress("supercool@dorky.com");
```

The toString() method

Every class can declare a special method which allows for getting a String representation of an object's state. The member could have such a method which could return a reference to a String containing the name and email of the instance for which the method was called. We'll explain this in more detail in the chapter about inheritance, but this is what such a method could look like:

```
public String toString(){
 return this.name + ", " + this.email;
}
```

A String representation for an instance could be e.g.:

Ada, ada@lovelace.org

Tying it together

```
public class Member{
  private String name;
  private String email;

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

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

  public String emailAddress(){
    return email;
  }

  public String name(){
    return name;
  }

  public void changeEmailAddress
                    (String email){
    this.email = email;
  }

  public String toString(){
    return name + ", " + email;
  }
}
```

Example test program

```
public class TestMember{
  public static void main(String[] args){
    Member onlyName = new Member("Ada");
    Member nameAndEmail = new Member("Eva", "eva@email.com");
    System.out.println(onlyName); // toString called magically?
    System.out.println(nameAndEmail);
    nameAndEmail.changeEmailAddress("eva@new-email.com");
    System.out.println(nameAndEmail);
    System.out.println( onlyName.name() );
  }
}
```

```
$ javac TestMember.java && java TestMember
Ada, null
Eva, eva@email.com
Eva, eva@new-email.com
Ada
```