PART 2

With the introduction of what an *object* is, now we are ready to learn the CONSTRUCTOR concept.

Just to refresh our memory, let's take a look at what we have learned in part 1.
A sample class declaration, as far as we have learned, looks like:

```
class Book
{
        String title;
        String author;
        int year;
}
```

And when we want to make use of this Book class, we do the following:

```
class Application
{
        public static void main(String args[])
        {
                Book book1 = new Book();

                book1.title = "The Secret";
                book1.year = 2006;
                book1.author = "Rhonda Byrne";
        }
}
```

Let's say in another application, we want to create more than one book.

Here is the code that creates two book objects and initializes the fields of each created object:

```
class Application
{
        public static void main(String args[])
        {
                Book book1 = new Book();  // Create the object

                book1.title = "The Secret";  // initalize the title field
                book1.year = 2006;  // initialize the year field
                book1.author = "Rhonda Byrne"; // initialize the author field

                Book book2 = new Book();

                book2.title = "Stuff on My Cat";
                book2.year = 2006;
                book2.author = "Mario Garza";
        }
}
```

As you may have already noticed, we have written 8 lines of code just for creating and initalizing two book objects. What if the Book class had more than 3 fields? Then, it would be even longer to create and initalize a book object.

Isn't there a easier and cleanier way to create & initalize the fields of objects?

YES, there is! It is done using CONSTRUCTORS!

A constructor is a perfect candidate for initializing the fields of an object. You may think of constructors as methods that only get executed when the object gets created. That means, it only gets executed when the "new" keyword is used.

In the previous examples, what we have done is:

- First create the object by the *Book book1 = new Book();* statement
- Then go ahead and initialize the fields of the book object

Constructors will allow us to initialize the fields **AS** we create the book object.

Enough said, everything will become much clearer with an example.

First let's add the constructor to our Book class declaration:

```
class Book
{
        String title;
        int year;
        String author;

        // The constructor!
        Book(String initialTitle, int initialYear, String initialAuthor)
        {
                title = initialTitle;
                year = initialYear;
                author = initialAuthor;
        }
}
```

With this class declaration in hand, here is how we can make use of the constructor (the explanation will follow):

```
class Application
{
        public static void main(String args[])
        {
                // All in one line!
                Book book1 = new Book("The Secret", 2006, " Rhonda Byrne");
        }
}
```

OK, now it is time to analyze what we have done.
First the constructor declaration:

```
Book(String initialTitle, int initialYear, String initialAuthor)
{
        title = initialTitle;
        year = initialYear;
        author = initialAuthor;
}
```

• The first thing you will notice is the way you define a constructor. The name of the constructor should be identical to the class name. Since our class name was Book, the constructor name also gets to be Book.
• The second thing is the parameters of the constructors. The Book class has three fields that may be initalized. In correspondance with the number of fields, the constructor also takes in 3 parameters that will be initalized to the fields of the class.
• The third thing is that the constructors do not have any return type. That is a good way to distinguish constructors from methods: constructors cannot have a return type.

Now, let's see how the constructor is used from an application. The following line is enough to execute the constructor:

```
Book book1 = new Book("The Secret", 2006, " Rhonda Byrne");
```

This single line says a lot.

i) First, create a brand new book object.

ii) And at the same time, make sure that the fields of the object are initialized to what I indicate in the parameters.

So, it must be fairly easy to guess the output of the following code:

```
class Application
{
        public static void main(String args[])
        {
                Book book1 = new Book("The Secret", 2006, " Rhonda Byrne");
                System.out.println(book1.title);
        }
}
```

The Secret

The constructor concept may have raised certain questions. Here are some sample questions and answers.

Q: What does actually happen when the

Book book1 = new Book("The Secret", 2006, " Rhonda Byrne");

line is executed?

A: Let's try to trace the code when the line gets executed.

1) Book book1 = new Book("The Secret", 2006, "Rhonda Byrne");
The new keyword here should remind you of a new object construction. Therefore, the constructor of the object will be executed.

2) Book(String initialTitle, int initialYear, String initialAuthor) // in the class declaration
The parameters of the constructors get their values. So,

initialTitle = "The Secret"
initialYear = 2006;
initialAuthor = "Rhonda Byrne"

3) title = initialTitle;
The title field of the object, gets the value of the initialTitle variable. Since initialTitle variable was previously set to "The Secret" in the second step, the title field of the object gets set to "The Secret".
4) year = initialYear; // Same idea
5) author = initialAuthor; // Same idea

Q: Now I remember one of the first examples that I had done. I was able to create a book object using the

Book book1 = new Book();

statement, but I am sure Book class didn't have any constructors or anything like that. How was that possible?

A: An excellent point! Here is the story behind that:

Java is a very clever language. For creating a new object, you definitely need a constructor. Knowing that, if you did not put in any constructor, Java inserts a default (no additional functionality) constructor for your object.

That means, when you did your exercise, your Book class declaration actually looked like:

```
class Book
{
        String title;
        int year;
        String author;

        // The default constructor
        Book()
        {
                // no effect at all
        }
}
```

Thanks to this default constructor, you were able to create a new Book object. However, obviously, this default constructor does not initalize any fields of the class. That was why previously you had to do it yourself.

Q: What about the number of constructors in a class?

A: There is no limit on the number of constructors of a class. You could have as many as you want.
Furthermore, it is a good idea to provide various options. For example, for the Book class you could also have these constructors:

```
class Book
{
        String title;
        int year;
        String author;

        Book(String initialTitle, int initialYear, String initialAuthor)
        {
                title = initialTitle;
                year = initialYear;
                author = initialAuthor;
        }

        Book(String initialTitle)
        {
                title = initialTitle;
                year = 0;
                author = "";
        }
}
```

In the class declaration, you will notice the second constructor. It only asks for the title of the book and initalizes the title. Other fields will be initialized to default values.

Here is how we may make use of the second constructor:

```
class Application
{
        public static void main(String args[])
        {
                // We don't know the author or the year of the book yet
                Book book1 = new Book("The Secret");
                ...
                // We now learn the author and the year of the book
                book1.year = 2006;
                book1.author = "Rhonda Byrne";
        }
}
```

Our new constructor gave us an option. We have created the book object, assuming that we had only known the title of the book in the beginning. Therefore, we have used the constructor that solely initalizes the title of the book.

As a review, let's make a comparison:

Before Constructors

```
class Application
{
      public static void main(String args[])
      {
          Book book1 = new Book();

          book1.title = "The Secret";
          book1.year = 2006;
          book1.author = "Rhonda Byrne";

          Book book2 = new Book();

          book2.title = "Stuff on My Cat";
          book2.year = 2006;
          book2.author = "Mario Garza";
       }
}
```

After Constructors

```
class Application
{
      public static void main(String args[])
      {
          Book book1 = new Book("The Secret", 2006, " Rhonda Byrne");
          Book book2 = new Book("Stuff on My Cat", 2006, "Mario Garza");
      }
}
```

We have come a long way! We know have a good idea of what an object is and furthermore we know how to make use of the constructors of objects.

The next 7-8 slides do not contain any new material. Instead, they are there to expand your vision about objects by working on a little more sophisticated problem.

Although it is relatively easy to grasp the notion of an object, one can easily get lost when examples get a little complicated. There is one specific example that seems to be the most confusing for the beginners: the idea of *composition*: having an object inside another object.

Knowing that, there will be an emphasis on this topic. The following slides will come up with examples in order to make sure that this example is well-understood.

So, what does it mean to have an object in an object? It would be best to explain this concept with an example.

Imagine that you are taking an introductory object orientation course and you have received a new assignment from your professor.

The assignment asks you to create a Person class. Here are the properties of the requested Person class:

- The name of the person,
- The age of the person,
- The job/occupation of the person
- The favorite book of the person

Furthermore, the assignment provides the Book class implementation that we have seen in many parts of the tutorial. The specifications require you to make use of this Book class in your implementation of the Person class.

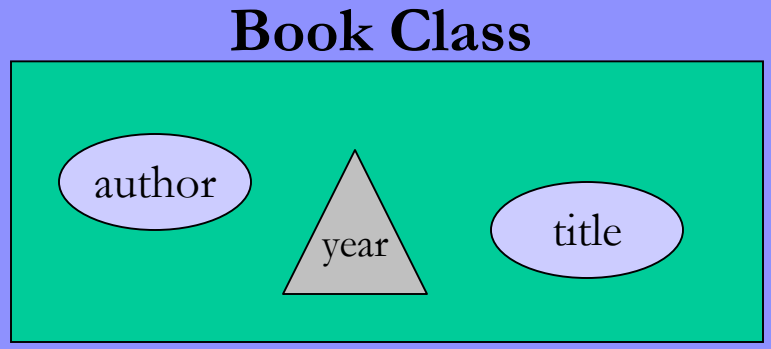
At first glance, the specifications may seem confusing. How can one make the connection between two different classes?

Let's start building our Person class without really thinking about the Book class implementation.
It would look like:

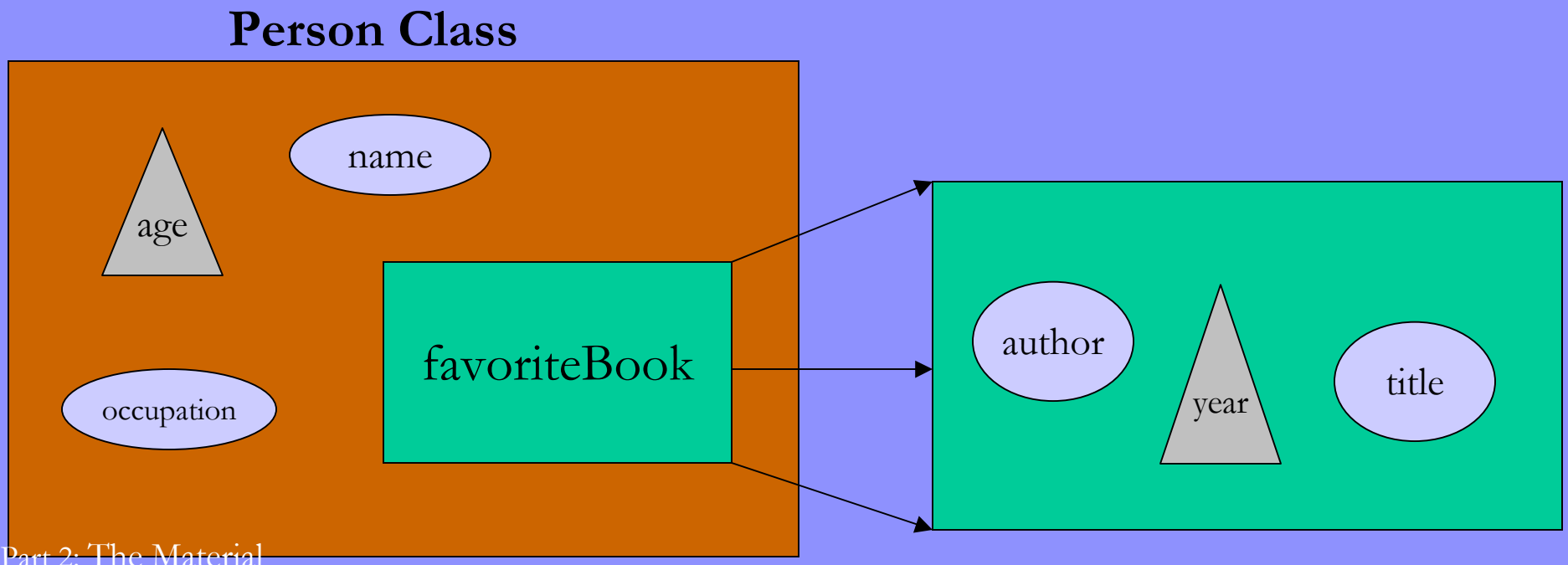
```
class Person
{
        String name;
        int age;
        String occupation;

         // the book part that we are not sure of yet
        // of what type should it be? Simply a String? Or something else?
        ??? favoriteBook;

}
```

At this point, referring back to the box analogy would be a great idea!

As a reminder, here is the box analogy that we have used in part 1:



Basically, the book class encapsulates three different information: title of the book, author of the book and the year of the book.
The same idea will apply for the Person class. This time the Person class will actually include a reference a book object (explanation will follow).



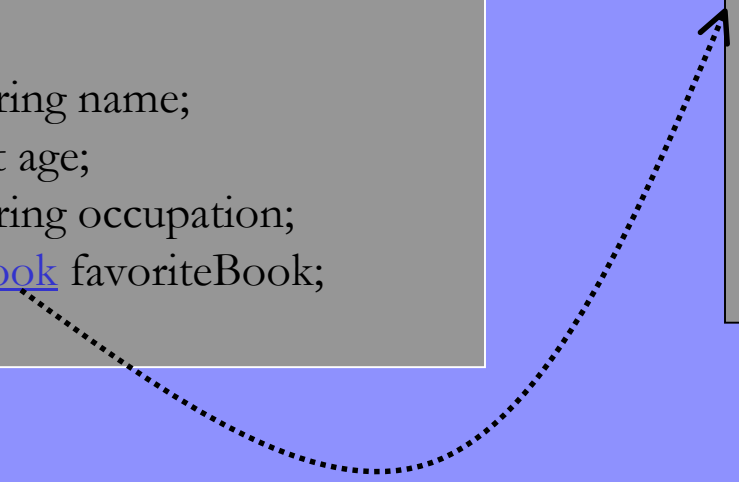
According to the figure, here is the hiearchy:

- Person
  - ✓ A name (String)
  - ✓ An occupation (String)
  - ✓ An age (int)
  - A favorite Book
    - ✓ A title (String)
    - ✓ An author (String)
    - ✓ The year (int)

With the figure and the hiearchy in hand, let's try to write the code for the Person class.

```
class Person
{
        String name;
        int age;
        String occupation;
        Book favoriteBook;
}
```

```
class Book
{
        String title;
        String author;
        int year;
}
```



Now, let's add constructors for each class so that our solution looks nice. For the Book class:

```
class Book
{
        String title;
        String author;
        int year;
        Book(String initialTitle, String initialAuthor, int initialYear)
        {
                title = initialTitle;
                author = initialAuthor;
                year = initialYear;
        }
}
```

```
class Person
{
        String name;
        String occupation;
        int age;
        Book favoriteBook;

        Person(String initialName, String initialOccupation, int initialAge, Book initialFavoriteBook)
        {
                name = initialName;
                occupation = initialOccupation;
                age = initialAge;
                favoriteBook = initialFavoriteBook;
        }
}
```

Until now, this was the most complicated code that we have written!
We know have two nice classes, let's use them in an application!

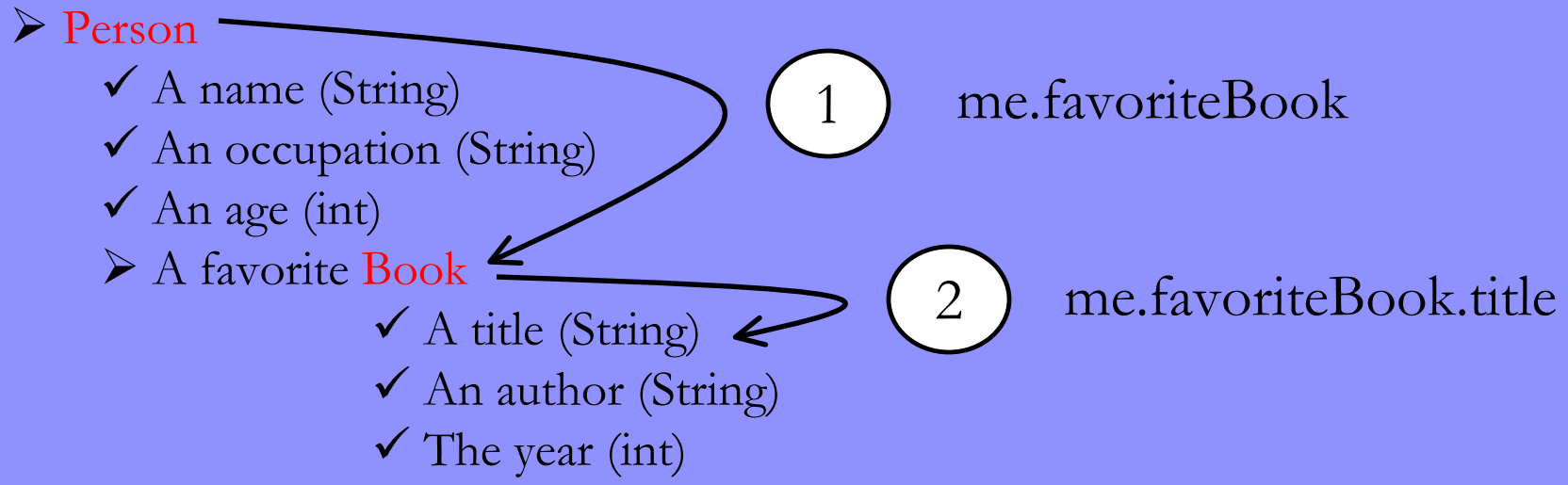
What about creating yourself using the Person class!
Here is an application that does that:

```
class Application
{
        public static void main(String args[])
        {
                Book book = new Book("The Secret", "Rhonda Byrne", 2006);

                Person me = new Person("Anıl","Student",18,book);

                System.out.println("My name is: " + me.name);
                System.out.println("I am " + me.age + " years old.");
                System.out.println("I am a " + me.occupation);
                System.out.println("The title of my favorite book is: " + me.favoriteBook.title);
        }
}
```

The last line says the following:

" The title of the favorite book of the person"

Let's use the following diagram to show how one can access the title of the favorite book of a person.



So, this was the whole idea behind the object inside an object concept.

There is one last minor detail that has to be covered, before finishing this second part of the tutorial.

Q: There were certain occasions in which we didn't want to attach any particular value to a variable/field. For example, we had a name field which referred to the name of a person. If we didn't know the name of the person, then we just typed ***name = " "*** to indicate that the name is unkown.

Now with the last example, we could have a case where we have a person but we don't really know the favorite book of that person. In that case, how can we state that the favorite book is unknown? Would ***favoriteBook = " "*** help?

A: The suggested syntax, unfortunately, won't be valid. Instead, we will make use of a brand new keyword for indicating that the favorite book is unknown.

The keyword is: ***null***

Let's again try to create another person. Let this person be a friend of yours. However this time, unfortunately, you don't really know the favorite book of your friend. How would you create your friend?

```
class Application
{
        public static void main(String args[])
        {
                Person myFriend = new Person("Anıl","Student",18,null);

                System.out.println("My name is: " + me.name);
                System.out.println("I am " + me.age + " years old.");
                System.out.println("I am a " + me.occupation);
        }
}
```

When you create your friend, instead of passing a book object to the constructor, you type null. That simply means: "I have no idea about the book, so I am giving you nothing".

Until now, we have made an introduction to object-oriented programming and we have made a lot progress on the *object* concept. Now, it is a good time to take a step back a little and take a look at the big picture.

- Why almost every software company in the world makes use of Object-Oriented programming?
- What makes Object-Oriented programming unique? How is it different than procedural programming? What capabilities does it provide?

By the end of this tutorial, you will have complete answers for these questions.
For right now, let's just start with one aspect of Object-Oriented programming: How it helps people to share or re-use the codes that they have written.

Let's imagine that your professor wants you to write a small application for the school library. The users of this application must be able to browse the books in the library and see if they are available or not.

Remember the book example from the first part of the tutorial? Here is the code for the Book class that we have written previously:

```
class Book
{
        String title;
        String author;
        int year;
}
```

In your library application, you will most probably need to store the title and author of the books in the library. Since you have already performed a similar task and created this Book class, it would be a very good idea to **re**-use this code.

Or, similarly, let's say that a friend of yours is asking your help. S/he is working on a project and s/he knows that you had been working on a library application. S/he is asking if you have previously written a Book class so that s/he can use it in his/her project.

If you agree to share your code, then all you have to do is send him/her the Book.java (the code below) class along with a simple documentation that describes your implementation of a Book.

```
class Book
{
        String title;
        String author;
        int year;
}
```

This is an example of how you could share your code with someone else. All your friend has to do right now is to add the Book.java class to his/her project, read its documentation and start using the Book object. There isn't even a need to look at your code!

Let's do some brainstorming and try to imagine what would happen if you didn't use Object-Oriented programming. Would you be able to share your code with your friend?

Without object-orientation your original code would most probably look like:

```
class Application
{
        public static void main(String args[])
        {
                String book1Title = "";
                String book1Author = "";
                int book1Year = 0;

                // more code here .....
        }
}
```

This code wouldn't be a good candidate for sharing. In order to make use of your code, your friend first has to analyze your code. Then, s/he has to understand how you have chosen to represent a book. And even after that, s/he will have to copy and paste those parts that she is interested in.

As a result, it could be concluded that procedural programming doesn't really support sharing / re-using of code in a nice way.

Of course, these were very simple examples that emphasize on how object-oriented programming facilitates code reuse and code sharing. Try to imagine how nice it would be if the shared code contained thousands of lines!

This is indeed how giant projects (for example Microsoft Office) are implemented!
Speficially, let's think about the spelling functionality provided by Microsoft Word. With the help of object-orientation, the same spelling functionality can be easily re-used in Microsoft Outlook to spell-check our e-mails.

Millions of people do programming. That means billions of lines of code are written. One capability of object-orientation is to increase the re-usability of these codes, so that better and bigger projects could be implemented.

- End of the Material for Part 2 -

Review of Concepts

- Constructors are useful structures that help to initialize the fields of an object.
- The constructor of an object is automatically called as the object gets created.
- A constructor can accept parameters, which may be used to initialize the fields of the object.
- In order to come up with more complicated data structures, an object may have another object as one of its field.
- Such a composition allows the construction of bigger and better classes.
- Object-oriented programming facilitates the code sharing among the people. An object created by one person, may easily be re-used by someone else when both people speak in terms of objects.
- The sharing of object-oriented code, supported by a good documentation, greatly helps the construction of large-scaled projects.

Review of Syntax

- Adding a constructor to a class declaration:

```
Classname(parameters)
{
    // your code goes here
}
```

1. Constructor name = Class name
2. No return type!

- When the value of an object is unknown, you may use the **null** keyword.

e.g. favoriteBook = null

Exercise

- We want you to model an employee, with the following properties:
  - An employee should have a name and surname
  - An employee should have a social insurance number
  - The birthdate of the employee should be stored for insurance purposes
  - Similarly, the system should also know that date that the employee is hired
- It is indicated that the dates are very important for this application. Therefore, the dates will be stored in detail with the following properties
  - The year
  - The month
  - The day

Exercise (continued)

- For each class that you have created, you should have appropriate constructors.

- To show that you are capable of the classes you have designed, create a very simple application.

- In the application do the following:

  1. Create an employee named Alan Smith.
  2. His birth date is 2nd of September 1974.
  3. His social insurance number is 231-5632-22.
  4. He has been hired by his company on the 8th of March 2005.

Solution

```
class Date
{
        // the fields of the Date class
        int year;
        int month;
        int day;

        // the constructor that initializes all the fields
        Date(int initialYear, int initialMonth, int initialDay)
        {
                year = initialYear;
                month = initialMonth;
                day = initialDay;
        }
}
```

Solution (continued)

```
class Employee
{
        // the fields of the Employee class
        String name;
        String surname;
        String insuranceNumber;
        Date birthDate;
        Date hireDate;

        // the constructor that initializes all the fields
        Employee(String initialName, String initialSurname, String initialNumber, Date
initialBirthDate, Date initialHireDate)
        {
                name = initialName;
                surname = initialSurname;
                insuranceNumber = initialNumber;
                birthDate = initialBirthDate;
                hireDate = initialHireDate;
        }
}
```

Solution (continued)

```
class Application
{
   public static void main(String args[])
   {
      Date birth  = new Date(1974,9,2);
      Date hiring = new Date(2005,3,8);

      Employee employee = new Employee("Alan", "Smith", "231-5632-22", birth, hiring)
   }
}
```