Pierre Ruiz
matriculation 08009004

Edinburgh Napier UNIVERSITY

Coursework 2 submission for Software Development 2 (SET08108 2016-7)

---

1) Source code: PersonComponent.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Abstract component class used to make sure that
     * components and decorators share the same specification
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public abstract class PersonComponent
    {
        /*
         * Must return the PersonComponent's name.
         */
        public abstract String Name { get; }

        /*
         * Returns the PersonComponent's address (null if the
         * PersonComponent is not a customer).
         */
        public virtual String GetAddress()
        {
            return null;
        }

        /*
         * Returns the PersonComponent's customer number (-1 if the
         * PersonComponent is not a customer).
         */
        public virtual int GetCustNb()
        {
            return -1;
        }
```

```
/*
 * Returns the PersonComponent's passport number (null if the
 * PersonComponent is not a customer).
 */
public virtual String GetPassportNb()
{
    return null;
}

/*
 * Returns the PersonComponent's age (-1 if the
 * PersonComponent is not a customer).
 */
public virtual int GetAge()
{
    return -1;
}

/*
 * Returns the PersonComponent itself; and references is null.
 */
public virtual PersonComponent Unwrap(
                                  out List<PersonDecorator> references)
{
    references = null;
    return this;
}

/*
 * Returns true if the PersonComponent wraps another
 * BookingDecorator, otherwise false.
 */
public virtual bool isDecorator()
{
    return false;
}

/*
 * Returns the PersonComponent itself.
 */
public virtual PersonComponent Undecorate(PersonDecorator reference)
{
    return this;
}

/*
 * Returns the PersonComponent itself.
 */
public virtual PersonComponent UndecorateOnce()
{
    return this;
}

/*
 * Must return a textual representation of the
 * PersonComponent in order to persist it to a CSV file.
 */
public abstract String ToCSV();
```

```
        /*
         * Returns false.
         */
        public virtual bool IsCustomer()
        {
            return false;
        }

        /*
         * Returns false.
         */
        public virtual bool IsGuest()
        {
            return false;
        }
    }
}
```

2) Source code: Person.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Represents individuals.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public class Person : PersonComponent
    {
        // PROPERTIES:

        private String name;
        public override String Name
        {
            get
            {
                return this.name;
            }
        }
```

```
        // METHODS:

        /*
         * Constructor.
         *
         * throws ArgumentException if firstName or secondName is null or
         * equals String.Empty
         */
        public Person(String name)
        {
            if (String.IsNullOrEmpty(name))
            {
                throw new ArgumentException("ConcretePerson.Name must not"
                                            + " be null nor String.Empty");
            }

            this.name = name;
        }

        /*
         * Returns a textual representation of the Person object
         * in order to persist it to a CSV file; fields must come
         * in the same order as enumerated in PersonFields.cs
         */
        public override String ToCSV()
        {
            return "#PERSON\r\n" + Name + "\r\n";
        }
    }
}
```

3) Source code: PersonDecorator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Abstract class defining the minimum implementation of
     * a concrete PersonDecorator.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public abstract class PersonDecorator : PersonComponent
    {
        // PROPERTIES:

        // the decorated component
        protected PersonComponent decoratedComponent;
        public void SetComponent(PersonComponent p)
        {
            decoratedComponent = p;
        }
```

```
// the PersonDecorator's name (is null if the root component is not
// a concrete Person)
public override string Name
{
    get
    {
        String name = null;

        if (decoratedComponent != null)
        {
            name = decoratedComponent.Name;
        }

        return name;
    }
}

// METHODS:

/*
 * Returns the PersonDecorator's Address (is null if the root
 * component is not a concrete Person)
 */
public override string GetAddress()
{
    String address = null;

    if (decoratedComponent != null)
        // && decoratedComponent.GetType() != typeof(Person))
    {
        address = decoratedComponent.GetAddress();
    }

    return address;
}

/*
 * Returns the PersonDecorator's customer number (-1 if the root
 * component is not a concrete Person)
 */
public override int GetCustNb()
{
    int custNb = -1;

    if (decoratedComponent != null)
    {
        custNb = decoratedComponent.GetCustNb();
    }

    return custNb;
}
```

```
/*
 * Returns the PersonDecorator's passport number (is null if the root
 * component is not a concrete Person)
 */
public override string GetPassportNb()
{
    String passportNb = null;

    if (decoratedComponent != null)
    // && decoratedComponent.GetType() != typeof(Person))
    {
        passportNb = decoratedComponent.GetAddress();
    }

    return passportNb;
}

/*
 * Returns the PersonDecorator's age (-1 if the root
 * component is not a concrete Person)
 */
public override int GetAge()
{
    int age = -1;

    if (decoratedComponent != null)
    {
        age = decoratedComponent.GetCustNb();
    }

    return age;
}

/*
 * Returns the root PersonComponent of the decoration stack;
 * references outputs a list of pointers to all the PersonDecorator
 * instances in the decoration stack.
 */
public override PersonComponent Unwrap(
                                    out List<PersonDecorator> references)
{
    PersonComponent component = this;
    List<PersonDecorator> decorators = new List<PersonDecorator>();

    while (component.isDecorator())
    {
        decorators.Add((PersonDecorator)component);
        component = ((PersonDecorator)component).decoratedComponent;
    }

    references = decorators;
    return component;
}
```

```
/*
 * Returns true if the PersonDecorator wraps another
 * PersonDecorator, otherwise false.
 */
public override bool isDecorator()
{
    return this.decoratedComponent != null;
}

/*
 * Returns a reference to a new PersonComponent identical in content
 * to calling instance, except unwraped from the PersonDecorator
 * passed as a parameter (or the to the Booking itself if it is not
 * decorated at all).
 */
public override PersonComponent Undecorate(PersonDecorator reference)
{
    if (this == reference) return decoratedComponent;
    /* Short circuit method if the decorator to remove is the
     * last one added.
     */
    List<PersonDecorator> references;
    PersonComponent result = this.Unwrap(out references);

    foreach (PersonDecorator decorator in references)
    {
        if (decorator != reference)
        {
            decorator.decoratedComponent = result;
            result = decorator;
        }
    }

    return result;
}

/*
 * Peels a single layer of decoration by returning the decorator's
 * decorated component.
 */
public override PersonComponent UndecorateOnce()
{
    if (decoratedComponent != null)
    {
        return decoratedComponent;
    }
    else
    {
        return this;
    }

}
```

```
        /*
         * Returns a textual representation of the decorated
         * PersonComponent in order to persist it to a CSV file.
         */
        public override String ToCSV()
        {
            String csv = String.Empty;

            if (decoratedComponent != null)
            {
                csv = decoratedComponent.ToCSV();
            }

            return csv;
        }

        /*
         * Returns true if the PersonComponent is a Customer, otherwise false.
         */
        public override bool IsCustomer()
        {
            bool isCustomer = false;

            if (decoratedComponent != null)
            {
                isCustomer = decoratedComponent.IsCustomer();
            }

            return isCustomer;
        }

        /*
         * Returns true if the PersonComponent is a Guest, otherwise false.
         */
        public override bool IsGuest()
        {
            bool isGuest = false;

            if (decoratedComponent != null)
            {
                isGuest = decoratedComponent.IsGuest();
            }

            return isGuest;
        }
    }
}
```

4) Source code: Customer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Represents a customer.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public class Customer : PersonDecorator
    {
        // PROPERTIES:

        private String address;
        public override String GetAddress()
        {
            return this.address;
        }

        private int customerNb;
        public override int GetCustNb()
        {
            return this.customerNb;
        }

        // METHODS:

        /*
         * Constructor.
         *
         * throws ArgumentException if customerRefNb is not strictly greater
         * than 0, or if address equals either String.Empty or null
         */
        public Customer(String address, int customerRefNb)
        {
            if (String.IsNullOrEmpty(address))
            {
                throw new ArgumentException("Customer.address"
                                            + " must not be null nor"
                                            + " String.Empty");
            }

            if (customerRefNb <= 0)
            {
                throw new ArgumentException("Customer.customerRefNb must be"
                                            + " strictly greater than 0");
            }

            this.address = address;
            this.customerNb = customerRefNb;
        }
```

```
        /*
         * Returns a textual representation of the decorated Person
         * (post Guest decoration) in order to persist it to a CSV file;
         * fields must come in the same order as enumerated in
         * CustomerFields.cs
         */
        public override String ToCSV()
        {
            return base.ToCSV() + "#CUSTOMER\r\n"
                                + customerNb
                                + ","
                                + address
                                + "\r\n";
        }

        /*
         * Returns true.
         */
        public override bool IsCustomer()
        {
            return true;
        }

        /*
         * Returns true if the Customer is also a Guest, otherwise flase
         */
        public override bool IsGuest()
        {
            return base.IsGuest();
        }
    }
}
```

5) Source code: Guest.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Represents a guest.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public class Guest : PersonDecorator
    {
        // PROPERTIES:

        private String passportNb;
        public override String GetPassportNb()
        {
            return this.passportNb;
        }
```

```
private int age;
public override int GetAge()
{
    return this.age;
}

// METHODS:

/*
* Constructor.
*
* throws ArgumentException if age is lesser than zero, or if
* passportNb equals String.Empty or null
*/
public Guest(String passportNb, int age)
{
    if (String.IsNullOrEmpty(passportNb))
    {
        throw new ArgumentException("ConcreteDecoratorGuest.passportNb"
                                    + " must not be null nor"
                                    + " String.Empty");
    }

    if (age < 0)
    {
        throw new ArgumentException("ConcreteDecoratorGuest.age must"
                                    + " not be lesser than 0");
    }

    this.passportNb = passportNb;
    this.age = age;
}

/*
 * Returns a textual representation of the decorated ConcretePerson
 * (post Guest decoration) in order to persist it to a CSV file;
 * fields must come in the same order as enumerated in GuestFields.cs
 */
public override String ToCSV()
{
    return base.ToCSV() + "#GUEST\r\n"
                        + passportNb
                        + ","
                        + age
                        + "\r\n";
}

/*
 * Returns true if the Guest is also a Customer, otherwise flase
 */
public override bool IsCustomer()
{
    return base.IsCustomer();
}
```

```
        /*
         * Returns true.
         */
        public override bool IsGuest()
        {
            return true;
        }
    }
}
```

6) Source code: PersonFactory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Singleton utility class, instantiates and decorates
     * PersonComponents.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public class PersonFactory
    {
        //Properties:
        private int nextCustNb;
        private static PersonFactory instance;
        public static PersonFactory Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new PersonFactory();
                }
                return instance;
            }
        }

        /*
         * Private constructor, to prevent class instantiation from
         * external classes (singleton class).
         */
        private PersonFactory()
        {
            this.nextCustNb = 1;
        }
```

```
/*
 * Restores the factory state according to sysData values.
 * Returns true if restore was successful, otherwise false.
 */
public bool Restore(Dictionary<String, String> sysData)
{
    String nxtBookNbString;
    bool outcome;
    outcome = sysData.TryGetValue(
                    BookingFactoryField.NEXT_BOOKING_NB.ToString(),
                    out nxtBookNbString);
    outcome = Int32.TryParse(nxtBookNbString, out this.nextCustNb);
    return outcome;
}

/*
 * Generates a new customer instance on the basis of the data
 * passed as parameters.
 */
public PersonComponent GetNewCustomer(String name, String address)
{
    PersonComponent p = new Person(name);
    PersonDecorator c = new Customer(address, this.nextCustNb);
    this.nextCustNb++;
    c.SetComponent(p);
    return c;
}

/*
 * Returns a reference to the updated PersonComponent instance.
 */
public PersonComponent UpdateCustomer(PersonComponent customer,
                                      String newName,
                                      String newAddress)
{
    // create a fresh customer with the new values but same customerNb
    PersonComponent p = new Person(newName);
    PersonDecorator c = new Customer(newAddress, customer.GetCustNb());
    c.SetComponent(p);

    // get all decorators of customer being updated
    List<PersonDecorator> decorators;
    customer.Unwrap(out decorators);

    // decorate the new customer with those decorators except those
    // of type 'Customer'
    foreach (PersonDecorator decorator in decorators)
    {
        if (decorator.GetType() != typeof(Customer))
        {
            decorator.SetComponent(c);
            c = decorator;
        }
    }

    return c;
}
```

```
/*
 * Instantiates a Customer from a dictonary<attribute, values>,
 * presumably recovered from persisted data (the dictonary keys
 * should follow the naming implemented in the *Field.cs enumerations)
 *
 * Thows Argument exceptions if there is a problem with the contents
 * of the dictionary passed as a parameter.
 */
public PersonDecorator RestoreCustomer(
                            Dictionary<String, String> attributes)
{
    String name;
    String address;
    String custRef;
    int custRefNb;
    String passportNb;
    String ageString;
    int age;

    if (!attributes.TryGetValue("NAME", out name)
     || !attributes.TryGetValue("ADDRESS", out address)
     || !attributes.TryGetValue("CUSTOMER_NUMBER", out custRef))
    {
        throw new ArgumentException("the attributes dictonary doesn't"
                                    + " have the expected keys.");
    }

    if (!Int32.TryParse(custRef, out custRefNb))
    {
        throw new ArgumentException("impossible to parse "
                                    + "[key: CUSTOMER_NUMBER, value: "
                                    + custRef + "] to type Int32.");
    }

    PersonDecorator c = new Customer(address, custRefNb);
    c.SetComponent(new Person(name));
    PersonDecorator g;

    if (attributes.TryGetValue("PASSPORT_NUMBER", out passportNb)
     && attributes.TryGetValue("AGE", out ageString))
    {
        if (!Int32.TryParse(ageString, out age))
        {
            throw new ArgumentException("impossible to parse "
                                        + "[key: AGE, "
                                        + " value: "+ ageString
                                        + "] to type Int32.");
        }
        g = new Guest(passportNb, age);
        g.SetComponent(c);
    }
    else
    {
        g = c;
    }

    return g;
}
```

```
/*
 * Generates a new guest instance on the basis of the data
 * passed as parameters.
 */
public PersonComponent GetNewGuest(String name,
                                   String passportNb,
                                   int age)
{
    Person p = new Person(name);
    Guest g = new Guest(passportNb, age);
    g.SetComponent(p);
    return g;
}

/*
 * Generates a new guest instance on the basis of the data
 * passed as parameters.
 */
public PersonComponent GetNewGuest(
                            PersonComponent customer,
                            String passportNb,
                            int age)
{
    Guest g = new Guest(passportNb, age);
    g.SetComponent(customer);
    return g;
}
```

```
        /*
         * Instantiates a Guest from a dictonary<attribute, values>,
         * presumably recovered from persisted data (the dictonary keys
         * should follow the naming implemented in the *Field.cs enumerations)
         *
         * Thows Argument exceptions if there is a problem with the contents
         * of the dictionary passed as a parameter.
         */
        public PersonComponent RestoreGuest(
                                    Dictionary<String, String> attributes)
        {
            String name;
            String passportNb;
            String ageString;
            int age;

            if (!attributes.TryGetValue("NAME", out name)
             || !attributes.TryGetValue("AGE", out ageString)
             || !attributes.TryGetValue("PASSPORT_NUMBER", out passportNb))
            {
                throw new ArgumentException("the attributes dictonary doesn't"
                                            + " have the expected keys.");
            }

            if (!Int32.TryParse(ageString, out age))
            {
                throw new ArgumentException("impossible to parse "
                                            + "[key: AGE, value: " + ageString
                                            + "] to type Int32.");
            }

            Guest g = new Guest(passportNb, age);
            g.SetComponent(new Person(name));

            return g;
        }

        /*
         * Returns a textual representation of the PersonFactory object
         * in order to persist it to a CSV file.
         */
        public String ToCSV()
        {
            return "#PERSON_FACTORY\r\n" + nextCustNb + "\r\n";
        }
    }
}
```

7) Source code: BookingComponent.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Abstract component class used to make sure that
     * components and decorators share the same specification
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public abstract class BookingComponent
    {
        /*
         * Must add a guest to a given BookingComponent.
         */
        public abstract void AddGuest(PersonComponent guest);

        /*
         * Must return the Booking number.
         */
        public abstract int GetBookingNb();

        /*
         * Must return the Booking's cutomer.
         */
        public abstract PersonComponent GetCustomer();

        /*
         * Must return the Booking's list of guests.
         */
        public abstract List<PersonComponent> GetGuests();

        /*
         * Must return the number of guests included in this
         * BookingComponent.
         */
        public abstract int GetNbGuests();

        /*
         * Must return the Booking start and end dates.
         */
        public abstract void GetDates(out DateTime arrival,
                                      out DateTime departure);

        /*
         * Must return the number of nights booked.
         */
        public abstract int GetNbNights();
```

```
        /*
         * Must return the the cost for each individual night booked.
         */
        public abstract float GetCostPerNight();

        /*
         * Returns the BookingComponent itself; and references is null.
         */
        public virtual BookingComponent Unwrap(
                                        out List<BookingDecorator> references)
        {
            references = null;
            return this;
        }

        /*
         * Returns true if the BookingDecorator wraps another
         * BookingDecorator, otherwise false.
         */
        public virtual bool isDecorator()
        {
            return false;
        }

        /*
         * Returns the BookingComponent itself.
         */
        public virtual BookingComponent Undecorate(BookingDecorator reference)
        {
            return this;
        }

        /*
         * Must return a textual representation of the
         * BookingComponent in order to persist it to a CSV file.
         */
        public abstract String ToCSV();
    }
}
```

8) Source code: Booking.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Represents a booking.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public class Booking : BookingComponent
    {
        // PROPERTIES:

        // the booking reference number.
        private int bookingNb;
        public override int GetBookingNb()
        {
            return this.bookingNb;
        }

        // the booking's customer.
        private PersonComponent cust;
        public override PersonComponent GetCustomer()
        {
            return this.cust;
        }

        // the booking start and end dates.
        private DateTime arrival;
        private DateTime departure;
        public override void GetDates(out DateTime arrival,
                                      out DateTime departure)
        {
            arrival = this.arrival;
            departure = this.departure;
        }

        //the booking's list of guests.
        private List<PersonComponent> guests = new List<PersonComponent>();
        public override List<PersonComponent> GetGuests()
        {
            return this.guests;
        }
```

```
// METHODS:

/*
 * Constructor.
 *
 * throws ArgumentException if bookingNb is not strictly greater
 * than 0, if departure day is not strictly later than arrival
 * day, or if customer is not decorated as a customer.
 */
public Booking(int bookingNb,
               PersonComponent customer,
               DateTime arrival,
               DateTime departure)
{
    if (customer.GetCustNb() <= 0)
    {
        throw new ArgumentException("Booking.cust must be"
                                    + "  decorated as a Customer");
    }

    if (bookingNb <= 0)
    {
        throw new ArgumentException("Booking.bookingNb must be"
                                    + " strictly greater than 0");
    }

    if (departure.DayOfYear <= arrival.DayOfYear
     && departure.Year <= arrival.Year)
    {
        throw new ArgumentException("departure day must be strictly"
                                    + " later than arrival day");
    }
    this.bookingNb = bookingNb;
    this.cust = customer;
    this.arrival = arrival;
    this.departure = departure;
}

/*
 * True if the contents of both bookings are identical, otherwise
 * false
 */
public override bool Equals(object obj)
{
    // self check
    if (this == obj) return true;

    // null check
    if (obj == null) return false;

    // type check and cast
    if (this.GetType() != obj.GetType()) return false;
    Booking b = (Booking)obj;

    // attribute content comparison
    return this.bookingNb == b.bookingNb
        && this.cust == b.cust
            // only checking reference equality because i am lacking
            // the time to implement customer 'deep' equallity check
        && this.arrival == b.arrival
        && this.departure == b.departure;
}
```

```
/*
 * Adds a guest to the booking.
 *
 * Throws ArgumentException if there is already 4 guests added to
 * the booking.
 */
public override void AddGuest(PersonComponent guest)
{
    if (guests.Count >= 4)
    {
        throw new ArgumentException("this booking already has"
                                    + " 4 guests");
    }
    this.guests.Add(guest);
}

/*
 * Returns the number of guests included in this Booking.
 */
public override int GetNbGuests()
{
    return guests.Count;
}

/*
 * Returns the number of nights booked.
 */
public override int GetNbNights()
{
    return (departure - arrival).Days;
}

/*
 * Returns the cost for each individual night booked.
 */
public override float GetCostPerNight()
{
    float costPerNight = 0;

    foreach (PersonComponent g in guests)
    {
        if (g.GetAge() < 18)
        {
            costPerNight += 30;
        }
        else
        {
            costPerNight += 50;
        }
    }

    return costPerNight;
}
```

```
        /*
         * Returns a textual representation of the Booking in order
         * to persist it to a CSV file.
         */
        public override String ToCSV()
        {
            int custIndex = indexOfCustomer();

            StringBuilder csvBooking = new StringBuilder("#BOOKING\r\n");
            csvBooking.Append(bookingNb + ",");
            csvBooking.Append(arrival.ToString().Substring(0, 10) + ",");
            csvBooking.Append(departure.ToString().Substring(0, 10) + "\r\n");

            if (custIndex >= 0)
            {
                csvBooking.Append(guests.ElementAt(custIndex).ToCSV());
            }
            else
            {
                csvBooking.Append(cust.ToCSV());
            }

            if (GetNbGuests() > 0)
            {
                foreach (Guest g in guests)
                {
                    if (guests.IndexOf(g) != custIndex)
                    {
                        csvBooking.Append(g.ToCSV());
                    }

                }
            }

            return csvBooking.ToString();
        }

        /*
         * Returns the index of the element that is also a customer in
         * list of guests, or -1 if none is.
         */
        private int indexOfCustomer()
        {
            int i = -1;

            foreach (Guest g in guests)
            {
                if (g.IsCustomer())
                {
                    i = guests.IndexOf(g);
                }
            }

            return i;
        }
    }
}
```

9) SourceCode: BookingDecorator.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

// temp
using System.Windows;

namespace Program
{
    /* Abstract class defining the minimum implementation of
     * a concrete BookingDecorator.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public abstract class BookingDecorator : BookingComponent
    {
        // PROPERTIES:

        // the BookingDecorator component
        protected BookingComponent decoratedComponent { get; set; }
        public void Setcomponent(BookingComponent booking)
        {
            decoratedComponent = booking;
        }

        // METHODS:

        /*
         * Adds a guest to the decorated booking.
         */
        public override void AddGuest(PersonComponent guest)
        {
            if (decoratedComponent != null)
            {
                decoratedComponent.AddGuest(guest);
            }
        }

        /*
         * Returns the decorated BookingComponent's list of booking number
         * (or -1 if the root component is not a concrete Booking).
         */
        public override int GetBookingNb()
        {
            int bookingNb = -1;

            if (decoratedComponent != null)
            {
                bookingNb = decoratedComponent.GetBookingNb();
            }

            return bookingNb;
        }
```

```
/*
 * Returns the decorated BookingComponent's customer
 * (or null if the root component is not a concrete Booking).
 */
public override PersonComponent GetCustomer()
{
    PersonComponent customer = null;

    if (decoratedComponent != null)
    {
        customer = decoratedComponent.GetCustomer();
    }

    return customer;
}

/*
 * Returns the decorated BookingComponent's list of guests
 * (or null if the root component is not a concrete Booking).
 */
public override List<PersonComponent> GetGuests()
{
    List<PersonComponent> guests = null;

    if (decoratedComponent != null)
    {
        guests = decoratedComponent.GetGuests();
    }

    return guests;
}

/*
 * Returns the root BookingComponent of the decoration stack;
 * references outputs a list of pointers to all the BookingDecorator
 * instances in the decoration stack.
 */
public override BookingComponent Unwrap(
                                  out List<BookingDecorator> references)
{
    BookingComponent component = this;
    List<BookingDecorator> decorators = new List<BookingDecorator>();

    while (component.isDecorator())
    {
        decorators.Add((BookingDecorator)component);
        component = ((BookingDecorator)component).decoratedComponent;
    }

    references = decorators;
    return component;
}
```

```
/*
 * Returns true if the BookingDecorator wraps another
 * BookingDecorator, otherwise false.
 */
public override bool isDecorator()
{
    return this.decoratedComponent != null;
}

/*
 * Returns a reference to a new BookingComponent identical in content
 * to calling instance, except unwraped from the BookingDecorator
 * passed as a parameter (or the to the Booking itself if it is not
 * decorated at all).
 */
public override BookingComponent Undecorate(BookingDecorator reference)
{
    if (this == reference) return decoratedComponent;
        /* Short circuit method if the decorator to remove is the
         * last one added.
         */
    List<BookingDecorator> references;
    BookingComponent result = this.Unwrap(out references);

    foreach (BookingDecorator decorator in references)
    {
        if (decorator != reference)
        {
            decorator.decoratedComponent = result;
            result = decorator;
        }
    }

    return result;
}

/*
 * Returns the booking start and end dates, or 01/01/1970 for both
 * if an error occurs.
 */
public override void GetDates(out DateTime arrival,
                              out DateTime departure)
{
    DateTime a = new DateTime(1970, 1, 1);
    DateTime d = new DateTime(1970, 1, 1);

    if (decoratedComponent != null)
    {
        decoratedComponent.GetDates(out a, out d);
    }

    arrival = a;
    departure = d;
}
```

```
/*
 * Returns the number of guests included in the decorated
 * BookingComponent (or -1 if the root component is not a
 * concrete Booking).
 */
public override int GetNbGuests()
{
    int nbGuests = -1;

    if (decoratedComponent != null)
    {
        nbGuests = decoratedComponent.GetNbGuests();
    }

    return nbGuests;
}

/*
 * Returns the number of nights included in the decorated
 * BookingComponent (or -1 if the root component is not a
 * concrete Booking).
 */
public override int GetNbNights()
{
    int nbNights = -1;

    if (decoratedComponent != null)
    {
        nbNights = decoratedComponent.GetNbNights();
    }

    return nbNights;
}

/*
 * Returns the cost for each individual night booked
 * (or -1 if the root component is not a concrete Booking).
 */
public override float GetCostPerNight()
{
    float costPerNight = -1;

    if (decoratedComponent != null)
    {
        costPerNight = decoratedComponent.GetCostPerNight();
    }

    return costPerNight;
}
```

```
        /*
         * Returns the cost of the decorated BookingComponent
         * (or -1 if the root component is not a concrete Booking).
         */
        public virtual float GetExtraCost()
        {
            float cost = -1;

            if (decoratedComponent != null
                && decoratedComponent.isDecorator())
            {
                cost = ((BookingDecorator)decoratedComponent).GetExtraCost();
            }

            return cost;
        }

        /*
         * Returns a textual representation of the decorated
         * BookingDecorator in order to persist it to a CSV file
         * (or String.Empty if the root component is not a
         * concrete Booking).
         */
        public override String ToCSV()
        {
            String csv = String.Empty;

            if (decoratedComponent != null)
            {
                csv = decoratedComponent.ToCSV();
            }

            return csv;
        }
    }
}
```

10)SourceCode: Breakfast.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Represents a breakfast extra.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public class Breakfast : BookingDecorator
    {
        // PROPERTIES:

        private String dietRequirement;
        public String GetDietRequirements()
        {
            return this.dietRequirement;
        }

        // METHODS:

        /*
         * Constructor.
         */
        public Breakfast(String dietRequirement)
        {
            this.dietRequirement = dietRequirement;
        }

        /*
         * Returns the extra cost of the breakfasts.
         */
        public override float GetExtraCost()
        {
            return 5 * base.decoratedComponent.GetNbGuests()
                     * base.decoratedComponent.GetNbNights();
        }

        /*
         * Returns a textual representation of the decorated
         * BookingDecorator in order to persist it to a CSV file.
         */
        public override String ToCSV()
        {
            return base.ToCSV() + "#BREAKFAST\r\n"
                                + dietRequirement
                                + "\r\n";
        }
    }
}
```

11)SourceCode: EveningMeal.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Represents an evening meals extra.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public class EveningMeal : BookingDecorator
    {
        // PROPERTIES:

        private String dietRequirement;
        public String GetDietRequirements()
        {
            return this.dietRequirement;
        }

        // METHODS :

        /*
         * Constructor.
         */
        public EveningMeal(String dietRequirement)
        {
            this.dietRequirement = dietRequirement;
        }

        /*
         * Returns the extra cost for the evening meals.
         */
        public override float GetExtraCost()
        {
            return 15 * base.decoratedComponent.GetNbGuests()
                      * base.decoratedComponent.GetNbNights();
        }

        /*
         * Returns a textual representation of the decorated
         * BookingDecorator in order to persist it to a CSV file.
         */
        public override String ToCSV()
        {
            return base.ToCSV() + "#EVENING_MEAL\r\n"
                                + dietRequirement
                                + "\r\n";
        }
    }
}
```

12)SourceCode: CarHire.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Represents a car hire extra.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public class CarHire : BookingDecorator
    {
        // PROPERTIES:

        private String driverName;
        public String GetDriverName()
        {
            return this.driverName;
        }

        private DateTime start;
        private DateTime end;
        public void GetHireDates(out DateTime start,
                                 out DateTime end)
        {
            start = this.start;
            end = this.end;
        }

        // METHODS:

        /*
         * Constructor.
         *
         * throws ArgumentException if (String.IsNullOrWhiteSpace(driverName)
         * or if end day is not strictly later than start day
         */
        public CarHire(String driverName, DateTime start, DateTime end)
        {
            if (String.IsNullOrWhiteSpace(driverName))
            {
                throw new ArgumentException("CarHire.driverName must not be"
                                            + " null, whitespace or an"
                                            + " empty string.");
            }

            if (end.DayOfYear <= start.DayOfYear
             && end.Year <= start.Year)
            {
                throw new ArgumentException("end day must be strictly"
                                            + " later than start day");
            }
            this.driverName = driverName;
            this.start = start;
            this.end = end;
        }
```

```
        /*
         * Returns the extra cost for the car hire.
         */
        public override float GetExtraCost()
        {
            return 50 * (end - start).Days;
        }

        /*
         * Returns a textual representation of the decorated
         * BookingDecorator in order to persist it to a CSV file.
         */
        public override String ToCSV()
        {
            StringBuilder csvCarHire = new StringBuilder("#CAR_HIRE\r\n");

            csvCarHire.Append(driverName + ",");
            csvCarHire.Append(start.ToString().Substring(0, 10) + ",");
            csvCarHire.Append(end.ToString().Substring(0, 10) + "\r\n");

            return base.ToCSV() + csvCarHire.ToString();
        }
    }
}
```

13)SourceCode: BookingFactory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Singleton utility class, instantiates and decorates
     * BookingComponents.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public class BookingFactory
    {
        //PROPERTIES:

        // the next booking unique number:
        private int nxtBookingNb;

        // points to the PersonFactory instance.
        private PersonFactory personFactory = PersonFactory.Instance;
```

```
// the singleton instance porperty
private static BookingFactory instance;
public static BookingFactory Instance
{
    get
    {
        if (instance == null)
        {
            instance = new BookingFactory();
        }
        return instance;
    }
}

// METHODS

/*
 * Private constructor, to prevent class instantiation from
 * external classes (singleton class).
 */
private BookingFactory()
{
    this.nxtBookingNb = 1;
}

/*
 * Restores the factory state according to sysData values.
 * Returns true if restore was successful, otherwise false.
 */
public bool Restore(Dictionary<String, String> sysData)
{
    String nxtBookNbString;
    bool outcome;
    outcome = sysData.TryGetValue(
                    PersonFactoryField.NEXT_CUST_NB.ToString(),
                    out nxtBookNbString);
    outcome = Int32.TryParse(nxtBookNbString, out this.nxtBookingNb);
    return outcome;
}

/*
 * Generates a new booking instance on the basis of the data
 * passed as parameters.
 */
public BookingComponent GetNewBooking(PersonComponent customer,
                                      DateTime arrival,
                                      DateTime departure)
{
    return new Booking(
                    this.nxtBookingNb++, customer, arrival, departure);
}
```

```
/*
 * Returns a reference to the updated booking instance.
 */
public BookingComponent UpdateBooking(int bookingNb,
                                      PersonComponent newCustomer,
                                      DateTime newArrival,
                                      DateTime newDeparture)
{
    return new Booking(
                    bookingNb, newCustomer, newArrival, newDeparture);
}

/*
 * Generates a new booking instance on the basis of the data
 * passed as parameters.
 */
private BookingComponent getNewBooking(int bookingNb,
                                       PersonComponent customer,
                                       DateTime arrival,
                                       DateTime departure)
{
    return new Booking(bookingNb, customer, arrival, departure);
}

/*
 * Decorates a BookingComponent, wrapping it inside a Breakfast
 * decorator.
 */
public Breakfast AddBreakfast(BookingComponent booking,
                              String dietRequirements)
{
    Breakfast bf = new Breakfast(dietRequirements);
    bf.Setcomponent(booking);
    return bf;
}

/*
 * Decorates a BookingComponent, wrapping it inside an EveningMeal
 * decorator.
 */
public EveningMeal AddEveningMeal(BookingComponent booking,
                                  String dietRequirements)
{
    EveningMeal em = new EveningMeal(dietRequirements);
    em.Setcomponent(booking);
    return em;
}
```

```
/*
 * Decorates a BookingComponent, wrapping it inside a CarHire
 * decorator.
 *
 * Throws ArgumentException if either  hire start or hire end
 * date falls without the booking dates range.
 */
public CarHire AddCarHire(BookingComponent booking,
                          String driverName,
                          DateTime start,
                          DateTime end)
{
    DateTime arrival;
    DateTime departure;
    booking.GetDates(out arrival, out departure);

    if (start < arrival || end > departure)
    {
        throw new ArgumentException("CarHire start & end dates must"
                                    + " be within the booking dates"
                                    + " range.");
    }

    CarHire ch = new CarHire(driverName, start, end);
    ch.Setcomponent(booking);
    return ch;
}
```

```
/*
 * Instantiates a BookingComponent from a
 * List<Dictonary<attribute, values>>, presumably recovered from
 * persisted data (the dictonaries keys should follow the naming
 * implemented in the *Field.cs enumerations)
 *
 * Thows Argument exception if there is a problem with the contents
 * of the dictionary passed as a parameter.
 */
public BookingComponent Restore(
                            List<Dictionary<String, String>> booking)
{
    BookingComponent result = null;

    Dictionary<String, String> bData;
    Dictionary<String, String> cData;
    List<Dictionary<String, String>> guestsData;
    extract(booking, out bData, out cData, out guestsData);

    String bookingNb;
    String csvArrival;
    String csvDeparture;

    if (bData != null
     && cData != null
     && bData.TryGetValue(BookingField.BOOKING_NUMBER.ToString(),
                          out bookingNb)
     && bData.TryGetValue(BookingField.ARRIVAL.ToString(),
                          out csvArrival)
     && bData.TryGetValue(BookingField.DEPARTURE.ToString(),
                          out csvDeparture))
    {
        result = getNewBooking(Int32.Parse(bookingNb),
                               personFactory.RestoreCustomer(cData),
                               Convert.ToDateTime(csvArrival),
                               Convert.ToDateTime(csvDeparture));

        if (result.GetCustomer().IsGuest())
        {
            result.AddGuest(result.GetCustomer());
        }
    }

    result = addGuestsData(result, guestsData);
    result = addExtrasData(result, bData);
    return result;
}
```

```
/*
 * Extracts the BookingComponent data, the Customer data and the
 * Guests data from contents of a dictonary<attribute, values>,
 * presumably recovered from persisteddata (the dictonary keys
 * should follow the naming implemented in the *Field.cs
 * enumerations)
 */
private void extract(
                List<Dictionary<String, String>> booking,
                out Dictionary<String, String> bookingData,
                out Dictionary<String, String> customerData,
                out List<Dictionary<String, String>> guestsData)
{
    Dictionary<String, String> bd = null;
    Dictionary<String, String> cd = null;
    List<Dictionary<String, String>> gd
                            = new List<Dictionary<String, String>>();

    foreach (Dictionary<String, String> d in booking)
    {
        if (d.ContainsKey(BookingField.BOOKING_NUMBER.ToString()))
        {
            bd = d;
        }
        else
        if (d.ContainsKey(CustomerField.CUSTOMER_NUMBER.ToString()))
        {
            cd = d;
        }
        else
        {
            gd.Add(d);
        }
    }
    bookingData = bd;
    customerData = cd;
    guestsData = gd;
}

/*
 * Adds Guests to a BookingComponent according to data contents of a
 * dictonary<attribute, values>, presumably recovered from persisted
 * data (the dictonary keys should follow the naming implemented in
 * the *Field.cs enumerations)
 */
private BookingComponent addGuestsData(
                            BookingComponent b,
                            List<Dictionary<String, String>> gData)
{
    foreach (Dictionary<String, String> d in gData)
    {
            b.AddGuest(personFactory.RestoreGuest(d));
    }
    return b;
}
```

```
/*
 * Decorates a BookingComponent according to data contents of a
 * dictonary<attribute, values>, presumably recovered from persisted
 * data (the dictonary keys should follow the naming implemented in
 * the *Field.cs enumerations)
 */
private BookingComponent addExtrasData(
                            BookingComponent b,
                            Dictionary<String, String> bData)
{
    if (b != null
     && bData != null)
    {
        String s1;
        String s2;
        String s3;

        if (bData.TryGetValue(
                        BreakfastField.DIET_REQUIREMENT_BREAKFAST
                                      .ToString(),
                        out s1))
        {
            b = AddBreakfast(b, s1);
        }

        if (bData.TryGetValue(
                        EveningMealField.DIET_REQUIREMENT_EVENING
                                        .ToString(),
                        out s1))
        {
            b = AddEveningMeal(b, s1);
        }

        if (bData.TryGetValue(CarHireField.DRIVER_NAME.ToString(),
                              out s1)
         && bData.TryGetValue(CarHireField.START.ToString(),
                              out s2)
         && bData.TryGetValue(CarHireField.END.ToString(),
                              out s3))
        {
            b = AddCarHire(b,
                           s1,
                           Convert.ToDateTime(s2),
                           Convert.ToDateTime(s3));
        }
    }
    return b;
}
```

```
        /*
         * Returns a textual representation of the BookingFactory object
         * in order to persist it to a CSV file.
         */
        public String ToCSV()
        {
            return "#BOOKING_FACTORY\r\n" + nxtBookingNb + "\r\n";
        }
    }
}
```

14)SourceCode: DataPersistenceFacade.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

// custom imports:
using System.IO;

namespace Program
{
    /*
     * Utility class, facade for the Data Persistence implementation
     * details.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public class DataPersistenceFacade
    {
        // PROPERTIES:

        // the bookings data persistence directory.
        private String dataDirectory = @"bookings";
        // the system data persistence directory.
        private String systemDirectory = @"system";
        // the data writer instance:
        private CSVWriter dataWriter = CSVWriter.Instance;
        // the data reader instance:
        private CSVReader dataReader = CSVReader.Instance;

        // METHODS RELATED TO BOOKINGS:

        /*
         * Persists the BookingComponent to {dataDirectory}/{bookingNb}.csv;
         * returns true if data was persisted successfuly or false if not.
         */
        public bool Persist(BookingComponent booking)
        {
            String filePath = (String.Format(@"{0}/{1}.csv",
                                             dataDirectory,
                                             booking.GetBookingNb()));

            return dataWriter.Persist(booking, filePath);
        }
```

```
/*
 * Builds a List<Dictionary<attribute, value>>, each representing
 * an entity of a given BookingComponent (the dictonary keys are named
 * as defined in the *Field.cs enumerations);
 * returns true if data was read successfuly, otherwise false.
 */
public bool Read(int bookingNb,
                 out List<Dictionary<String, String>> bookingData)
{
    String filePath = (String.Format(@"{0}/{1}.csv",
                                     dataDirectory,
                                     bookingNb));

    List<Dictionary<String, String>> results;
    bool wasSuccessful = dataReader.ReadBooking(filePath, out results);

    bookingData = results;
    return wasSuccessful;
}

/*
 * Returns a list of all booking numbers ever persisted.
 */
public List<int> GetAllBookingNbs()
{
    if (!Directory.Exists(dataDirectory))
    {
        dataWriter.CreateDir(dataDirectory);
    }

    String[] filePaths = Directory.GetFiles(dataDirectory);
    List<int> bookingNbs = new List<int>();

    foreach (String fp in filePaths)
    {
        bookingNbs.Add(extractBookingNb(fp));
    }

    return bookingNbs;
}

/*
 * Returns the booking number from given persisted booking file name.
 */
private int extractBookingNb(String bookingFileName)
{
    int start = bookingFileName.LastIndexOf("\\") + 1;
    int end = bookingFileName.IndexOf(".") - start;
    int bookingNb = -1;
    if (end > 0)
    {
        bookingNb = Int32.Parse(bookingFileName.Substring(start, end));
    }
    return bookingNb;
}
```

```
/*
 * Deletes the file storing given booking from disc if it
 * exists.
 */
public void Delete(int bookingNb)
{
    String filePath = (String.Format(@"{0}/{1}.csv",
                                     dataDirectory,
                                     bookingNb));

    if (File.Exists(filePath))
    {
        File.Delete(filePath);
    }
}

// METHODS RELATED TO CUSTOMERS:

/*
 * Returns a list of all booking numbers that were made by
 * a given customer.
 */
public List<int> GetAllBookingNbs(int customerNb)
{
    List<int> bookingNbs = new List<int>();
    String[] bookingFiles = new String[0];
    List<Dictionary<String, String>> bookingData;
    String value;

    if (Directory.Exists(dataDirectory))
    {
        bookingFiles = Directory.GetFiles(dataDirectory);
    }

    foreach (String fPath in bookingFiles)
    {
        if (dataReader.ReadBooking(fPath, out bookingData))
        {
            foreach (Dictionary<String, String> d in bookingData)
            {
                if (d.TryGetValue(CustomerField.CUSTOMER_NUMBER
                                                .ToString(),
                                  out value)
                 && Int32.Parse(value) == customerNb)
                {
                    bookingNbs.Add(extractBookingNb(fPath));
                }
            }
        }
    }

    return bookingNbs;
}
```

```
/*
 * Returns a list of all the customer numbers persisted to file.
 */
public List<int> GetAllCustomerNbs()
{
    int customerNb;

    if (!Directory.Exists(dataDirectory))
    {
        dataWriter.CreateDir(dataDirectory);
    }

    List<int> customerNbs = new List<int>();

    foreach (String bookingFile in Directory.GetFiles(dataDirectory))
    {
        customerNb = extractCustomerNb(bookingFile);
        if (!customerNbs.Contains(customerNb))
        {
            customerNbs.Add(customerNb);
        }
    }

    return customerNbs;
}

/*
 * Returns the customer number from given persisted booking file,
 * or -1 if the file cant be found.
 */
private int extractCustomerNb(String bookingFile)
{
    List<Dictionary<String, String>> bookingData;
    if (File.Exists(bookingFile)
     && dataReader.ReadBooking(bookingFile, out bookingData))
    {
        int customerNb;
        String customerNbString;

        foreach (Dictionary<String, String> entity in bookingData)
        {
            if (entity.TryGetValue(CustomerField.CUSTOMER_NUMBER
                                                .ToString(),
                                   out customerNbString)
             && Int32.TryParse(customerNbString, out customerNb))
            {
                return customerNb;
                /* short circuit looping and exit method as soon
                 * as customer number was found.
                 */
            }
        }
    }

    return -1; // or return -1
}
```

```
/*
 * Returns a Dictionary<attribute, value>, representing a Customer.cs
 * instance (the dictonary keys are named as defined in the *Field.cs
 * enumerations);
 * returns true if data was read successfuly otherwise false.
 */
public bool Read(int customerNb,
                 out Dictionary<String, String> customerData)
{
    Dictionary<String, String> result = null;
    List<int> bookings = GetAllBookingNbs(customerNb);
    bool wasSuccessful = false;

    if (bookings != null && bookings.Count > 0)
    {
        int bookingNb = bookings.ElementAt(0);
        List<Dictionary<String, String>> bookingData;
        String value;

        if (this.Read(bookingNb, out bookingData))
        {
            foreach (Dictionary<String, String> d in bookingData)
            {
                if (d.TryGetValue(CustomerField.CUSTOMER_NUMBER
                                                .ToString(),
                                  out value)
                 && Int32.Parse(value) == customerNb)
                {
                    result = d;
                    wasSuccessful = true;
                }
            }
        }
    }

    customerData = result;
    return wasSuccessful;
}

// METHODS RELATED TO SYSTEM STATE:

/*
 * Restores system state from file.
 */
public bool PersistSystemState()
{
    String personFactoryPath = (String.Format(@"{0}/{1}.csv",
                                    systemDirectory,
                                    "person-factory"));

    String bookingFactoryPath = (String.Format(@"{0}/{1}.csv",
                                    systemDirectory,
                                    "booking-factory"));

    return dataWriter.Persist(PersonFactory.Instance,
                              personFactoryPath)
        && dataWriter.Persist(BookingFactory.Instance,
                              bookingFactoryPath);
}
```

```
        /*
         * Returns a Dictionary<attribute, value>, representing the last
         * system state saved (the dictonary keys are named as defined in the
         * SystemField.cs enumeration);
         * returns true if data was read successfuly otherwise false.
         */
        public bool ReadSystemState(out Dictionary<String, String> sysData)
        {
            return dataReader.ReadSystemState(systemDirectory, out sysData);
        }
    }
}
```

15)SourceCode: CSVWriter.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

// Custom imports:
using System.IO;

namespace Program
{
    /*
     * Singleton utility class, persists booking data to CSV files.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public class CSVWriter
    {
        // PROPERTIES:

        // the singleton instance property
        private static CSVWriter instance;
        public static CSVWriter Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new CSVWriter();
                }
                return instance;
            }
        }

        // METHODS

        /*
         * Private constructor, to prevent class instantiation from
         * external classes (singleton class).
         */
        private CSVWriter() { }
```

```
/*
 * Persists the BookingComponent to given filePath, returns true if
 * data was persisted successfuly or false if not.
 */
public bool Persist(BookingComponent booking, String filePath)
{
    String dataDirName = Path.GetDirectoryName(filePath);

    if (!Directory.Exists(dataDirName))
    {
        CreateDir(dataDirName);
    }

    if (File.Exists(filePath))
    {
        File.Delete(filePath);
    }

    try
    {
        System.IO.File.AppendAllText(filePath, booking.ToCSV());
    }
    catch
    {
        return false;
    }

    return true;
}

public void CreateDir(String dirPath)
{
    Directory.CreateDirectory(dirPath);
}
```

```
        /*
         * Persists the PersonFactory to given filePath, returns true if
         * data was persisted successfuly or false if not.
         */
        public bool Persist(PersonFactory pf, String filePath)
        {
            String dataDirName = Path.GetDirectoryName(filePath);

            if (!Directory.Exists(dataDirName))
            {
                Directory.CreateDirectory(dataDirName);
            }

            if (File.Exists(filePath))
            {
                File.Delete(filePath);
            }

            try
            {
                System.IO.File.AppendAllText(filePath, pf.ToCSV());
            }
            catch
            {
                return false;
            }

            return true;
        }

        /*
         * Persists the PersonFactory to given filePath, returns true if
         * data was persisted successfuly or false if not.
         */
        public bool Persist(BookingFactory bf, String filePath)
        {
            String dataDirName = Path.GetDirectoryName(filePath);

            if (!Directory.Exists(dataDirName))
            {
                Directory.CreateDirectory(dataDirName);
            }

            if (File.Exists(filePath))
            {
                File.Delete(filePath);
            }

            try
            {
                System.IO.File.AppendAllText(filePath, bf.ToCSV());
            }
            catch
            {
                return false;
            }

            return true;
        }

    }
}
```

16)SourceCode: CSVReader.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

// Custom imports:
using System.IO;

namespace Program
{
    /*
     * Singleton utility class, reads bookings data from CSV files.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public class CSVReader
    {
        // PROPERTIES:

        // the singleton instance property
        private static CSVReader instance;
        public static CSVReader Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new CSVReader();
                }
                return instance;
            }
        }

        // METHODS

        /*
         * Private constructor, to prevent class instantiation from
         * external classes (singleton class).
         */
        private CSVReader() { }
```

```
/*
 * Outputs a Dictionary<String, String> data of the last system state
 * persisted (the dictonary keys naming follows the naming implemented
 * in the *Field.cs enumerations).
 * Returns true if data was recovered successfuly, otherwise false.
 */
public bool ReadSystemState(String sysDirectory,
                            out Dictionary<String, String> sysData)
{
    bool wasSuccessful = true;
    List<String> sysFilesLines = new List<String>();
    Dictionary<String, String> data = new Dictionary<string,string>();
    String[] tmp1;
    String[] tmp2;

    try
    {
        foreach (String file in Directory.GetFiles(sysDirectory))
        {
            sysFilesLines.AddRange(readLines(file));
        }

        for (int i = 0; i < sysFilesLines.Count; i += 2)
        {
            tmp1 = new String[2];
            tmp2 = sysFilesLines.ElementAt(i + 1).Split(',');
            tmp1[1] = tmp2[0];

            if (sysFilesLines.ElementAt(i)
                             .Equals("#PERSON_FACTORY"))
            {
                data = join(data, index<PersonFactoryField>(tmp1));
            }
            else if (sysFilesLines.ElementAt(i)
                                  .Equals("#BOOKING_FACTORY"))
            {
                data = join(data, index<BookingFactoryField>(tmp1));
            }
        }
    }
    catch
    {
        wasSuccessful = false;
    }


    sysData = data;
    return wasSuccessful;
}
```

```
/*
 * Outputs a List<Dictionary<String, String>>, each instance of which
 * stores an entity of a BookingComponent as <attribute, value> (the
 * dictonary keys follow the naming implemented in the *Field.cs
 * enumerations).
 * Returns true if data was recovered successfuly, otherwise false.
 */
public bool ReadBooking(String filename,
                        out List<Dictionary<String, String>> keysVals)
{
    bool wasSuccessful = true;
    List<String[]> extractedEntities;
    List<Dictionary<String, String>> indexedEntities = null;

    try
    {
        extractedEntities = extractClasses(readLines(filename));
        indexedEntities = new List<Dictionary<String, String>>();

        foreach (String[] sArr in extractedEntities)
        {
            indexedEntities.Add(index(sArr));
        }
    }
    catch
    {
        wasSuccessful = false;
    }

    keysVals = indexedEntities;
    return wasSuccessful;
}
```

```
/*
 * Reads from a CSV file and returns a list of strings, each
 * corresponding to a line from the file.
 *
 * Throws ArgumentException if the number of lines in the file was not
 * even, as per the CSV formating done within classes.
 */
private List<String> readLines(String filename)
    /*
     * RESOURCE:
     * https://msdn.microsoft.com/en-us/library/db5x7c0d(v=vs.110).aspx
     */
{
    List<String> csvLines = new List<String>();
    String line;
    StreamReader sr = new StreamReader(filename);

    line = sr.ReadLine();
    while (line != null)
    {
        csvLines.Add(line);
        line = sr.ReadLine();
    }
    sr.Close();

    if (csvLines.Count % 2 != 0)
    {
        throw new ArgumentException("Invalid CSV file: should contain"
                                    + " an even number of text"
                                    + " lines");
    }

    return csvLines;
}
```

```
/*
 * Extracts the different entities from a csv BookingComponent,
 * each representing the data for a different class of that
 * BookingComponent.
 */
private List<String[]> extractClasses(List<String> csvBooking)
{
    List<String[]> csvEntities = new List<String[]>();
    String[] booking = null;
    String[] person = null;

    for (int i = 0; i < csvBooking.Count; i = i+2 )
    {
        switch (csvBooking.ElementAt(i))
        {
            case "#BOOKING":
                if (person != null)
                {
                    csvEntities.Add(person);
                    person = null;
                }
                booking = new String[1] { csvBooking.ElementAt(i) };
                booking = append(booking, csvBooking.ElementAt(i + 1)
                                                    .Split(','));
                break;
            case "#BREAKFAST":
            case "#EVENING_MEAL":
            case "#CAR_HIRE":
                Array.Resize<String>(ref booking, booking.Length + 1);
                booking[booking.Length - 1] = csvBooking.ElementAt(i);
                booking = append(booking, csvBooking.ElementAt(i + 1)
                                                    .Split(','));
                break;
            case "#PERSON":
                if (person != null)
                {
                    csvEntities.Add(person);
                    person = null;
                }
                person = new String[1] { csvBooking.ElementAt(i) };
                person = append(person, csvBooking.ElementAt(i + 1)
                                                  .Split(','));
                break;
            case "#CUSTOMER":
            case "#GUEST":
                Array.Resize<String>(ref person, person.Length + 1);
                person[person.Length - 1] = csvBooking.ElementAt(i);
                person = append(person, csvBooking.ElementAt(i + 1)
                                                  .Split(','));
                break;
        }
    }

    csvEntities.Add(booking);
    csvEntities.Add(person);

    return csvEntities;
}
```

```
/*
 * Appends arr2 at the end of arr1.
 */
private String[] append(String[] arr1, String[] arr2)
/*
 * RESOURCES:
 * http://stackoverflow.com/questions/59217/merging-two-arrays-in-net
 * https://msdn.microsoft.com/en-us/library/system.array.copy(v=vs.110).aspx
 */
{
    int appendFrom = arr1.Length;
    Array.Resize<String>(ref arr1, arr1.Length + arr2.Length);
    Array.Copy(arr2, 0, arr1, appendFrom, arr2.Length);
    return arr1;
}
```

```
/*
 * Indexes a csv entity (String[]) into a
 * dictionary<attribute, value>.
 */
private Dictionary<String, String> index(String[] entity)
{
    List<String[]> dividedEntity = divide(entity);
    Dictionary<String, String> indexedEntity = null;

    foreach (String[] sArr in dividedEntity)
    {
        switch (sArr[0])
        {
            case "#BOOKING":
                indexedEntity = index<BookingField>(sArr);
                break;
            case "#BREAKFAST":
                indexedEntity = join(indexedEntity,
                                     index<BreakfastField>(sArr));
                break;
            case "#EVENING_MEAL":
                indexedEntity = join(indexedEntity,
                                     index<EveningMealField>(sArr));
                break;
            case "#CAR_HIRE":
                indexedEntity = join(indexedEntity,
                                     index<CarHireField>(sArr));
                break;

            case "#PERSON":
                indexedEntity = index<PersonField>(sArr);
                break;
            case "#CUSTOMER":
                indexedEntity = join(indexedEntity,
                                     index<CustomerField>(sArr));
                break;
            case "#GUEST":
                indexedEntity = join(indexedEntity,
                                     index<GuestField>(sArr));
                break;

            default:
                throw new ArgumentException("Can't index entity, "
                                    + " content of sArr[0] must be"
                                    + " either #BOOKING or #PERSON");
        }
    }
    return indexedEntity;
}
```

```
/*
 * Divides an entity (String[]) into a List<String[]>, each
 * element of which represents a Component class (from a
 * decorator pattern).
 */
private List<String[]> divide(String[] entity)
{
    List<String[]> dividedEntity = new List<String[]>();
    String[] section = new String[0];

    foreach (String s in entity)
    {
        if (s.StartsWith("#"))
        {
            if (section.Length > 0)
            {
                dividedEntity.Add(section);
            }
            section = new String[1];
            section[0] = s;
        }
        else
        {
            Array.Resize<String>(ref section, section.Length + 1);
            section[section.Length-1] = s;
        }
    }

    dividedEntity.Add(section);
    return dividedEntity;
}

/*
 * Indexes part of an entity's attributes (= section) into a
 * dictionary<attribute, value>.
 */
private Dictionary<String, String> index<T>(String[] entitySection)
{
    T[] keysArr = (T[])Enum.GetValues(typeof(T));
    Dictionary<String, String> indexedSection
                                    = new Dictionary<String, String>();
    foreach (T k in keysArr)
    {
        indexedSection.Add(
                    k.ToString(),
                    entitySection[Array.IndexOf(keysArr, k) + 1]);
    }

    return indexedSection;
}
```

```
        /*
         * Joins two Dictionary<String, String> (Union operation).
         */
        private Dictionary<String, String>
            join(Dictionary<String, String> d1, Dictionary<String, String> d2)
                /*
                 * RESOURCE:
                 * http://stackoverflow.com/questions/59217/merging-two-arrays-in-net
                 */
        {
            return d1.Union(d2)
                     .ToDictionary(k => k.Key, v => v.Value);
        }
    }
}
```

17)SourceCode: PersonField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Enumeration of Person.cs fields in the order they appear in
     * Person.ToCSV and the persisted file.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public enum PersonField { NAME }
}
```

18)SourceCode: CustomerField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Enumeration of Customer.cs fields in the order they appear in
     * Customer.ToCSV and the persisted file.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public enum CustomerField { CUSTOMER_NUMBER, ADDRESS }
}
```

19)SourceCode: GuestField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Enumeration of Guest.cs fields in the order they appear in
     * Guest.ToCSV and the persisted file.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public enum GuestField { PASSPORT_NUMBER, AGE }
}
```

20)SourceCode: PersonFactoryField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Enumeration of PersonFactory.cs fields in the order they
     * appear in PersonFactory.ToCSV() and the persisted file.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public enum PersonFactoryField { NEXT_CUST_NB }
}
```

21)SourceCode: BookingField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Enumeration of Booking.cs fields in the order they appear in
     * Booking.ToCSV() and the persisted file, excluding the fields
     * defined in PersonField.cs, CustomerField.cs & GuestField.cs
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public enum BookingField { BOOKING_NUMBER, ARRIVAL, DEPARTURE  }
}
```

22)SourceCode: BreakfastField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Enumeration of Breafast.cs fields in the order they appear in
     * Breakfast.ToCSV() and the persisted file.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public enum BreakfastField { DIET_REQUIREMENT_BREAKFAST }
}
```

23)SourceCode: EveningMealField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Enumeration of EveningMeal.cs fields in the order they appear in
     * EveningMeal.ToCSV() and the persisted file.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public enum EveningMealField { DIET_REQUIREMENT_EVENING }
}
```

24)SourceCode: CarHireField.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Enumeration of CarHire.cs fields in the order they appear in
     * CarHire.ToCSV() and the persisted file.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public enum CarHireField { DRIVER_NAME, START, END }
}
```

25)SourceCode: BookingFactory.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Enumeration of BOookingFactory.cs fields in the order they
     * appear in BookingFactory.ToCSV() and the persisted file.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public enum BookingFactoryField { NEXT_BOOKING_NB }
}
```

26)SourceCode: ModelFacade.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Program
{
    /*
     * Facade providing an interface to the system.
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-09
     */
    public class ModelFacade
    {
        // PROPERTIES:

        // points to the BookingFactory instance:
        private BookingFactory bFact;

        // ponts to the PersonFactory instance:
        private PersonFactory pFact;

        // points to a DataPersistenceFacade object:
        private DataPersistenceFacade dpFacade;

        // points to the BookingComponent current working instance:
        public BookingComponent CurrentBook { get; set; }

        // points to the PersonComponent current working instance:
        private PersonComponent CurrentCust { get; set; }
```

```
// METHODS:

/*
 * Constructor.
 */
public ModelFacade()

{
    bFact = BookingFactory.Instance;
    pFact = PersonFactory.Instance;
    dpFacade = new DataPersistenceFacade();

    Dictionary<String, String> sysData;
    if (dpFacade.ReadSystemState(out sysData))
    {
        bFact.Restore(sysData);
        pFact.Restore(sysData);
    }
}

/*
 * Recovers last system state persisted to file.
 */
public bool RestoreSystemSavedState()
{
    return false;
}

/*
 * Persists current system state to file.
 */
public bool PersistSystemState()
{
    return dpFacade.PersistSystemState();
}

/*
 * Returns a list of all the booking numbers persisted on disc.
 */
public List<int> GetAllBookingNbs()
{
    return dpFacade.GetAllBookingNbs();
}

/*
 * Returns a list of all the booking numbers persisted on disc made
 * by a given customer.
 */
public List<int> GetAllBookingNbs(PersonComponent customer)
{
    return dpFacade.GetAllBookingNbs(customer.GetCustNb());
}
```

```
/*
 * Returns a list of all the customer numbers persisted on disc.
 */
public List<int> GetAllCustomerNbs()
{
    return dpFacade.GetAllCustomerNbs();
}

public void DeleteBooking(int bookingNb)
{
    dpFacade.Delete(bookingNb);
}


// METHODS RELATED TO CURRENT BOOKING:

/*
 * True if there is a booking loaded in the system when method is
 * called, otherwise false.
 */
public bool IsABookingLoaded()
{
    return CurrentBook != null;
}

/*
 * Instanciates a new booking for the current customer.
 */
public void CreateBooking(DateTime arrival, DateTime departure)
{
    CurrentBook = bFact.GetNewBooking(CurrentCust, arrival, departure);
}

/*
 * Loads the booking matching given booking number into the system
 * (from persisted data).
 * Returns true if the booking was found & loaded successfully,
 * otherwise false.
 */
public bool RestoreBooking(int bookingNb)
{
    bool wasRestored = true;
    List<Dictionary<String, String>> bookingData;
    if (!dpFacade.Read(bookingNb, out bookingData))
    {
        wasRestored = false;
    }
    else
    {
        CurrentBook = bFact.Restore(bookingData);
        CurrentCust = CurrentBook.GetCustomer();
    }
    return wasRestored;
}
```

```
/*
 * Returns the current booking's booking number, or -1 if no booking
 * is currently loaded.
 */
public int GetCurrentBookNb()
{
    int currentBookingNb = -1;
    if (CurrentBook != null)
    {
        currentBookingNb = CurrentBook.GetBookingNb();
    }
    return currentBookingNb;
}

/*
 * Returns the current booking's departure and arrival dates.
 */
public void GetCurrentBookDates(out DateTime arrival,
                                out DateTime departure)
{
    CurrentBook.GetDates(out arrival, out departure);
}

/*
 * Returns a list of the names of all the guests currently booked for
 * the current booking.
 */
public List<String> GetGuestNames()
{
    List<String> guestNames = new List<String>();
    foreach (PersonComponent g in CurrentBook.GetGuests())
    {
        guestNames.Add(g.Name);
    }
    return guestNames;
}

/*
 * Returns the number of guests currently booked for the current
 * booking(also 0 if no booking is currently loaded).
 */
public int GetCurrentNbGuests()
{
    int nbGuests = 0;
    if (CurrentBook != null)
    {
        nbGuests = CurrentBook.GetNbGuests();
    }
    return nbGuests;
}
```

```
/*
 * Returns the current booking's number of nights, or -1 if no
 * booking is currently loaded.
 */
public int GetCurrentNbNights()
{
    int nbNights = -1;
    if (CurrentBook != null)
    {
        List<BookingDecorator> extras;
        nbNights = CurrentBook.Unwrap(out extras).GetNbNights();
    }
    return nbNights;
}

/*
 * Returns the current booking's cost per night, or -1 if no booking
 * is currently loaded.
 */
public float GetCurrentCostPerNight()
{
    float costPerNight = -1;
    if (CurrentBook != null)
    {
        List<BookingDecorator> extras = GetCurrentExtras();
        costPerNight = CurrentBook.Unwrap(out extras).GetCostPerNight();
    }
    return costPerNight;
}

/*
 * Updates the BookingComponent instance currently loaded in the
 * system.
 */
public void UpdateBooking(DateTime arrival, DateTime departure)
{
    List<PersonComponent> savedGuests = CurrentBook.GetGuests();
    List<BookingDecorator> decorationStack;
    BookingComponent booking = CurrentBook.Unwrap(out decorationStack);

    booking = bFact.UpdateBooking(booking.GetBookingNb(),
                                  CurrentCust,
                                  arrival,
                                  departure);

    if (decorationStack != null)
    {
        foreach (BookingDecorator reference in decorationStack)
        {
            reference.Setcomponent(booking);
            booking = reference;
        }
    }

    CurrentBook = booking;

    foreach (PersonComponent g in savedGuests)
    {
        CurrentBook.AddGuest(g);
    }
}
```

```
/*
 * Closes the BookingComponent instance currently loaded in the
 * system.
 */
public void CurrentBookingClose()
{
    CurrentBook = null;
    CurrentCust = null;
}

/*
 * Persists the BookingComponent instance currently loaded in the
 * system to file.
 * Returns true if the booking was saved successfully, otherwise
 * false.
 */
public bool PersistCurrentBooking()
{
    return dpFacade.Persist(CurrentBook);
}

// METHODS RELATED TO THE CURRENT CUSTOMER:

/*
 * True if a customer is currently loaded in the system, otherwise
 * false
 */
public bool IsACustomerLoaded()
{
    return CurrentCust != null;
}

/*
 * Instanciates a new customer.
 */
public void CreateCustomer(String name, String address)
{
    CurrentCust = pFact.GetNewCustomer(name, address);
}

/*
 * Loads the customer matching given customer number into the system
 * (from persisted data).
 * Returns true if the customer was found & loaded successfully,
 * otherwise false.
 */
public bool RestoreCustomer(int customerNb)
{
    bool wasRestored = true;
    Dictionary<String, String> customerData;
    if (!dpFacade.Read(customerNb, out customerData))
    {
        wasRestored = false;
    }
    else
    {
        CurrentCust = pFact.RestoreCustomer(customerData);
    }
    return wasRestored;
}
```

```
/*
 * Returns the current booking's customer number, or -1 if no booking
 * is currently loaded.
 */
public int GetCurrentCustNb()
{
    int customerNb = -1;
    if (CurrentCust != null)
    {
        customerNb = CurrentCust.GetCustNb();
    }
    return customerNb;
}

/*
 * Returns the current booking's customer name, or null if no booking
 * is currently loaded.
 */
public String GetCurrentCustName()
{
    String customerName = null;
    if (CurrentCust != null)
    {
        customerName = CurrentCust.Name;
    }
    return customerName;
}

/*
 * Returns the current booking's customer address, or null if no
 * booking is currently loaded.
 */
public String GetCurrentCustAdress()
{
    String customerAddress = null;
    if (CurrentCust != null)
    {
        customerAddress = CurrentCust.GetAddress();
    }
    return customerAddress;
}
```

```
/*
 * Updates given customer's details with new values for all bookings
 * of his.
 */
public void UpdateCurrentCustomer(String newName, String newAddress)
{
    BookingComponent processedBooking;
    List<PersonComponent> savedGuests;
    List<BookingDecorator> decorationStack;
    List<Dictionary<String, String>> bookingData;
    DateTime arrival;
    DateTime departure;

    // update values within all persisted bookings made by current
    // customer
    foreach (int bookingNb
             in dpFacade.GetAllBookingNbs(CurrentCust.GetCustNb()))
    {
        dpFacade.Read(bookingNb, out bookingData);

        processedBooking = bFact.Restore(bookingData)
                                .Unwrap(out decorationStack);

        savedGuests = processedBooking.GetGuests();

        processedBooking.GetDates(out arrival, out departure);

        processedBooking = bFact.UpdateBooking(
                                    processedBooking.GetBookingNb(),
                                    pFact.UpdateCustomer(
                                        processedBooking.GetCustomer(),
                                        newName,
                                        newAddress),
                                    arrival,
                                    departure);

        if (decorationStack != null)
        {
            foreach (BookingDecorator reference in decorationStack)
            {
                reference.Setcomponent(processedBooking);
                processedBooking = reference;
            }
        }

        foreach (PersonComponent g in savedGuests)
        {
            processedBooking.AddGuest(g);
        }

        CurrentCust = processedBooking.GetCustomer();
        dpFacade.Persist(processedBooking);
    }

    // reload current booking into the system to upload changes
    if (IsABookingLoaded())
    {
        RestoreBooking(CurrentBook.GetBookingNb());
    }
}
```

```
// METHODS RELATED TO CURRENT BOOKING'S GUESTS:

/*
 * Adds a new person to current booking's list of guests.
 */
public void AddGuest(String name, String passportNb, int age)
{
    CurrentBook.AddGuest(pFact.GetNewGuest(name, passportNb, age));
}

/*
 * Updates details of guest at given index in current booking's
 * list of guests.
 */
public void EditGuest(int index,
                      String name,
                      String passportNb,
                      int age)
{
    CurrentBook.GetGuests().RemoveAt(index);
    CurrentBook.GetGuests().Insert(index,
                                   pFact.GetNewGuest(name,
                                                     passportNb,
                                                     age));
}

/*
 * True if the element at given index in list of guests is a
 * cutomer.
 */
public bool IsGuestACustomer(int index)
{
    if (index >= 0)
    {
        return CurrentBook.GetGuests().ElementAt(index).IsCustomer();
    }
    else
    {
        return false;
    }
}
```

```
/*
 * Adds current customer to current booking's list of guests.
 */
public void AddCustomerToGuests(String passportNb, int age)
{
    List<PersonComponent> savedGuests = CurrentBook.GetGuests();
    DateTime arrival;
    DateTime departure;
    CurrentBook.GetDates(out arrival, out departure);

    CurrentCust = pFact.GetNewGuest(CurrentCust,
                                    passportNb,
                                    age);
    CurrentBook = bFact.UpdateBooking(CurrentBook.GetBookingNb(),
                                      CurrentCust,
                                      arrival,
                                      departure);

    foreach (PersonComponent g in savedGuests)
    {
        CurrentBook.GetGuests().Add(g);
    }

    CurrentBook.AddGuest(CurrentCust);
}

/*
 * Updates guest details of current customer at given index in current
 * booking's list of guests.
 */
public void EditCustomerGuestDetails(int index,
                                      String passportNb,
                                      int age)
{
    CurrentBook.GetGuests().RemoveAt(index);
    CurrentBook.GetGuests().Insert(index,
                                   pFact.GetNewGuest(
                                      CurrentCust.UndecorateOnce(),
                                      passportNb,
                                      age));
}
```

```
/*
 * Deletes the guest at given index in list of guests for the
 * current booking.
 * Undecorates that guest if if it is also a
 * customer and updates CurrentCustomer with the correct
 * memory reference.
 */
public void DeleteGuest(int index)
{
    // First unwrap guest decorator from current customer if they are
    // the guest being deleted:
    if (CurrentBook.GetGuests().ElementAt(index).IsCustomer())
    {
        List<PersonComponent> savedGuests = CurrentBook.GetGuests();
        DateTime arrival;
        DateTime departure;
        CurrentBook.GetDates(out arrival, out departure);

        CurrentCust = CurrentCust.UndecorateOnce();
        CurrentBook = bFact.UpdateBooking(CurrentBook.GetBookingNb(),
                                          CurrentCust,
                                          arrival,
                                          departure);

        foreach (PersonComponent g in savedGuests)
        {
            CurrentBook.GetGuests().Add(g);
        }


    }

    // Then delete selected guest reference from guests list:
    CurrentBook.GetGuests().RemoveAt(index);
}

// METHODS RELATED TO CURRENT BOOKING'S EXTRAS:

/*
 * Decorates the current booking with a Breakfast extra.
 */
public void AddBreakFast(String dietRequirements)
{
    CurrentBook = bFact.AddBreakfast(CurrentBook, dietRequirements);
}

/*
 * Decorates the current booking with an EveningMeal extra.
 */
public void AddEveningMeal(String dietRequirements)
{
    CurrentBook = bFact.AddEveningMeal(CurrentBook, dietRequirements);
}
```

```
/*
 * Decorates the current booking with a CarHire extra.
 */
public void AddCarHire(String driverName,
                       DateTime start,
                       DateTime end)
{
    CurrentBook = bFact.AddCarHire(CurrentBook,
                                   driverName,
                                   start,
                                   end);
}

/*
 * Updates properties of a given Breakfast.
 */
public void UpdateBreakfast(BookingDecorator reference,
                            String newDietRequirements)
{
    CurrentBook = CurrentBook.Undecorate(reference);
    CurrentBook = bFact.AddBreakfast(CurrentBook,
                                     newDietRequirements);
}

/*
 * Updates properties of a given EveningMeal.
 */
public void UpdateEveningMeal(BookingDecorator reference,
                              String newDietRequirements)
{
    CurrentBook = CurrentBook.Undecorate(reference);
    CurrentBook = bFact.AddEveningMeal(CurrentBook,
                                       newDietRequirements);
}

/*
 * Updates properties of a given CarHire.
 */
public void UpdateCarHire(BookingDecorator reference,
                          String newDriverName,
                          DateTime newStart,
                          DateTime newEnd)
{
    CurrentBook = CurrentBook.Undecorate(reference);
    CurrentBook = bFact.AddCarHire(CurrentBook,
                                   newDriverName,
                                   newStart,
                                   newEnd);
}

/*
 * Returns a list of references to each of the CurrentBooking's
 * decorators (= extras), or null if it is not decorated at all.
 */
public List<BookingDecorator> GetCurrentExtras()
{

    List<BookingDecorator> references;
    CurrentBook.Unwrap(out references);
    return references;
}
```

```
/*
 * Returns a list of textual representations of each of the
 * CurrentBooking's decorators (= extra).
 */
public List<String> GetCurrentExtrasNames()
{

    List<BookingDecorator> references = GetCurrentExtras();
    List<String> extras = new List<String>();

    if (references != null)
    {
        foreach (BookingDecorator bd in references)
        {
            if (bd.GetType() == typeof(EveningMeal))
            {
                extras.Add("Evening meal");
            }
            else if (bd.GetType() == typeof(Breakfast))
            {
                extras.Add("Breakfast");
            }
            else if (bd.GetType() == typeof(CarHire))
            {
                extras.Add("CarHire");
            }
        }
    }
    return  extras;
}

/*
 * Returns the current booking's cost for, breakfasts or -1 if no
 * booking is currently loaded.
 */
public float GetCurrentBreakfastCost()
{
    float breakfastsCost = -1;

    if (CurrentBook != null)
    {
        List<BookingDecorator> extras = GetCurrentExtras();

        breakfastsCost = 0;

        if (extras != null)
        {
            foreach (BookingDecorator e in extras)
            {
                if (e.GetType() == typeof(Breakfast))
                {
                    breakfastsCost += e.GetExtraCost();
                }
            }
        }
    }
    return breakfastsCost;
}
```

```
/*
 * Returns the current booking's cost for, evening meals or -1 if no
 * booking is currently loaded.
 */
public float GetCurrentEveningMealsCost()
{
    float eveningMealsCost = -1;

    if (CurrentBook != null)
    {
        List<BookingDecorator> extras = GetCurrentExtras();

        eveningMealsCost = 0;

        if (extras != null)
        {
            foreach (BookingDecorator e in extras)
            {
                if (e.GetType() == typeof(EveningMeal))
                {
                    eveningMealsCost += e.GetExtraCost();
                }
            }
        }
    }
    return eveningMealsCost;
}

/*
 * Returns the current booking's cost for, evening meals or -1 if no
 * booking is currently loaded.
 */
public float GetCurrentCarHireCost()
{
    float carHiresCost = -1;

    if (CurrentBook != null)
    {
        List<BookingDecorator> extras = GetCurrentExtras();

        carHiresCost = 0;

        if (extras != null)
        {
            foreach (BookingDecorator e in extras)
            {
                if (e.GetType() == typeof(CarHire))
                {
                    carHiresCost += e.GetExtraCost();
                }
            }
        }
    }
    return carHiresCost;
}
```

```
        /*
         * Removes the selected extra from the booking.
         */
        public void RemoveExtra(int index)
        {
            List<BookingDecorator> references = GetCurrentExtras();
            if (references != null)
            {
                CurrentBook = CurrentBook.Undecorate(
                                                references.ElementAt(index));
            }
        }
    }
}
```

27)SourceCode: MainWindow.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;


namespace Program
{
    /*
     * Interaction logic for MainWindow.xaml
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-09
     */
    public partial class MainWindow : Window
    {
        // PROPERTIES:

        // reference to a ModelFacade instance:
        private ModelFacade mFacade;

        // METHODS:

        /*
         * Constructor.
         */
        public MainWindow()
        {
            this.mFacade = new ModelFacade();
            InitializeComponent();
            clearDisplay();
        }
```

```
/*
 * Loads and displays the booking referenced by txtBookingRef.Text.
 */
private void btnLoadBooking_Click(object sender, RoutedEventArgs e)
{
    new WindowLoadBooking(mFacade).ShowDialog();
    refreshDisplay();
}

/*
 * Refreshes all fields displayed in the window according to
 * current system objects states.
 */
private void refreshDisplay()
{
    if (mFacade.IsABookingLoaded())
    {
        clearDisplay();
        refreshBookingDisplay();
        refreshCustomerDisplay();
        refreshGuestsDisplay();
    }
}

/*
 * Refreshes the booking fields displayed in the window.
 */
private void refreshBookingDisplay()
{
    DateTime start;
    DateTime end;
    mFacade.GetCurrentBookDates(out start, out end);

    // update labels content:
    lblBookingNbValue.Content = mFacade.GetCurrentBookNb().ToString();
    lblArrivalValue.Content = start.ToString().Substring(0, 10);
    lblDepartureValue.Content = end.ToString().Substring(0, 10);

    // make labels visible:
    lblBooking.Visibility = Visibility.Visible;
    lblBookingNb.Visibility = Visibility.Visible;
    lblBookingNbValue.Visibility = Visibility.Visible;
    lblArrival.Visibility = Visibility.Visible;
    lblArrivalValue.Visibility = Visibility.Visible;
    lblDeparture.Visibility = Visibility.Visible;
    lblDepartureValue.Visibility = Visibility.Visible;
}
```

```
/*
 * Refreshes the customer fields displayed in the window.
 */
private void refreshCustomerDisplay()
{
    // update labels content:
    lblCustomerNameValue.Content = mFacade.GetCurrentCustName();
    lblCustomerNbValue.Content = mFacade.GetCurrentCustNb().ToString();

    // make labels visible:
    lblCustomer.Visibility = Visibility.Visible;
    lblCustomerNb.Visibility = Visibility.Visible;
    lblCustomerNbValue.Visibility = Visibility.Visible;
    lblCustomerName.Visibility = Visibility.Visible;
    lblCustomerNameValue.Visibility = Visibility.Visible;
}

/*
 * Refreshes the guests fields displayed in the window.
 */
private void refreshGuestsDisplay()
{
    // update listbox content:
    foreach (String g in mFacade.GetGuestNames())
    {
        lstGuests.Items.Add(g);
    }

    // make label & list box visible:
    lblGuests.Visibility = Visibility.Visible;
    lstGuests.Visibility = Visibility.Visible;
}

/*
 * Empties all boxes displayed in the MainWindow.
 */
private void clearDisplay()
{
    // hide booking data display:
    lblBookingNbValue.Visibility = Visibility.Hidden;
    lblArrivalValue.Visibility = Visibility.Hidden;
    lblDepartureValue.Visibility = Visibility.Hidden;

    // hide customer data display:
    lblCustomerNbValue.Visibility = Visibility.Hidden;
    lblCustomerNameValue.Visibility = Visibility.Hidden;

    // hide guests data display:
    lstGuests.Items.Clear();
}
```

```
        /*
         * Closes the current booking.
         */
        private void btnCloseBooking_Click(object sender, RoutedEventArgs e)
        {
            mFacade.CurrentBookingClose();
            clearDisplay();
        }

        /*
         * Opens a WindowCreateEdit dialog to view & edit current booking
         * or create a new one.
         */
        private void btnNewEdit_Click(object sender, RoutedEventArgs e)
        {
            new WindowCreateEdit(mFacade).ShowDialog();
            refreshDisplay();
        }

        /*
         * Opens a WindowInvoice dialog to view current booking invoice.
         */
        private void btnInvoice_Click(object sender, RoutedEventArgs e)
        {
            if (!mFacade.IsABookingLoaded())
            {
                MessageBox.Show("There is no booking loaded in the system"
                                + " yet.\r\n"
                                + "Load a booking to view it's invoice.");
            }
            else
            {
                new WindowInvoice(mFacade).ShowDialog();
            }
        }

        /*
         * Saves the current system state when the program is closed.
         */
        private void Window_Closing(object sender,
                                    System.ComponentModel.CancelEventArgs e)
        {
            mFacade.PersistSystemState();
            MessageBox.Show("Cheerio");
        }

        /*
         * Deletes the booking curently loaded in the system.
         */
        private void btnDelete_Click(object sender, RoutedEventArgs e)
        {
            int bookingNb = mFacade.GetCurrentBookNb();
            mFacade.CurrentBookingClose();
            mFacade.DeleteBooking(bookingNb);
            clearDisplay();
        }
    }
}
```

28)SourceCode: WindowLoadBooking.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Program
{
    /*
     * Interaction logic for WindowLoadBooking.xaml
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-07
     */
    public partial class WindowLoadBooking : Window
    {
        // PROPERTIES:

        // points to a ModelFacade instance.
        private ModelFacade mFacade;

        // METHODS:

        /*
         * Constructor.
         */
        public WindowLoadBooking(ModelFacade mFacade)
        {
            this.mFacade = mFacade;
            InitializeComponent();
            refreshDisplay();
        }

        /*
         * Refreshes all fields displayed in the window according to
         * current system objects states.
         */
        private void refreshDisplay()
        {
            clearDisplay();
            lstBookings.ItemsSource = mFacade.GetAllBookingNbs();
        }
```

```
/*
 * Refreshes all booking detail fields displayed in the window
 * according to current booking state.
 */
private void refreshBookDetailDisplay()
{

    if (mFacade.IsABookingLoaded())
    {
        // update field contents:
        DateTime start;
        DateTime end;
        mFacade.GetCurrentBookDates(out start, out end);
        lblArrivalValue.Content = start.ToString().Substring(0, 10);
        lblDepartureValue.Content = end.ToString().Substring(0, 10);
        lblCustNameValue.Content = mFacade.GetCurrentCustName();
        lstGuests.Items.Clear();
        foreach (String g in mFacade.GetGuestNames())
        {
            lstGuests.Items.Add(g);
        }

        // make labels visible:
        lblArrivalValue.Visibility = Visibility.Visible;
        lblDepartureValue.Visibility = Visibility.Visible;
        lblCustNameValue.Visibility = Visibility.Visible;
    }
    else
    {
        refreshDisplay();
    }
}

/*
 * Empties all boxes displayed in the MainWindow.
 */
private void clearDisplay()
{
    clearBookingDetails();

    lstBookings.ItemsSource = null;
    lstGuests.Items.Clear();
}

/*
 * Hide all booking detail fields from the window.
 */
private void clearBookingDetails()
{
    lblArrivalValue.Visibility = Visibility.Hidden;
    lblCustNameValue.Visibility = Visibility.Hidden;
    lblDepartureValue.Visibility = Visibility.Hidden;
    lstGuests.Items.Clear();
}
```

```
        /*
         * loads selected booking into the system when double clicking on it.
         */
        private void lstBookings_MouseDoubleClick(object sender,
                                                  MouseButtonEventArgs e)
        {
            List<int> l = (List<int>) lstBookings.ItemsSource;
            int i = lstBookings.SelectedIndex;

            if (i < 0 || !mFacade.RestoreBooking(l.ElementAt(i)))
            {
                MessageBox.Show("Please double click on a booking from the"
                            + " list to load it into the system.");
            }

            refreshBookDetailDisplay();
        }
    }
}
```

29)SourceCode: WindowCreateEdit.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Program
{
    /*
     * Interaction logic for WindowCreateEdit.xaml
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-08
     */
    public partial class WindowCreateEdit : Window
    {
        //PROPERTIES:

        // reference to calling window's ModelFacade instance:
        private ModelFacade mFacade;
```

```
// METHODS:

/*
 * The window constructor.
 */
public WindowCreateEdit(ModelFacade modelFacade)
{
    this.mFacade = modelFacade;

    InitializeComponent();
    lblBookingRef.Content = "Booking number\r\n";

    if (!mFacade.IsABookingLoaded())
    {
        clearDisplay();
    }
    else
    {
        lblBookingRef.Content += mFacade.GetCurrentBookNb()
                                         .ToString();
        refreshDisplay();
    }
}

/*
 * Checks if the booking can be saved or updated on the basis of
 * window field values, and acts accordingly (closes the window,
 * except if the booking was a new one).
 */
private void btnSave_Click(object sender, RoutedEventArgs e)
{
    if (areAllValuesValid())
    {
        if (!mFacade.IsABookingLoaded())
        {
            mFacade.CreateBooking((DateTime)dtpArrival.SelectedDate,
                                  (DateTime)dtpDeparture.SelectedDate);
            mFacade.PersistCurrentBooking();
            refreshDisplay();
        }
        else
        {
            mFacade.UpdateBooking((DateTime)dtpArrival.SelectedDate,
                                  (DateTime)dtpDeparture.SelectedDate);
            mFacade.PersistCurrentBooking();
        }
    }
}
```

```
/*
 * True if all the window fields are valid to create or amend a
 * booking, otherwise false.
 * Displays error message windows.
 */
private bool areAllValuesValid()
{
    bool areValidValues = true;

    if (!mFacade.IsACustomerLoaded())
    {
        areValidValues = false;
        MessageBox.Show("Please create or load a customer for"
                        + " the booking before saving.");
    }

    if (dtpArrival.SelectedDate == null)
    {
        areValidValues = false;
        MessageBox.Show("Please select a valid arrival date"
                        + " before saving the booking.");
    }
    else if (dtpDeparture.SelectedDate == null)
    {
        areValidValues = false;
        MessageBox.Show("Please select a valid departure date"
                        + " before saving the booking.");
    }
    else if (dtpDeparture.SelectedDate <= dtpArrival.SelectedDate)
    {
        areValidValues = false;
        MessageBox.Show("Departure date must be strictly"
                        + " later than arrival date.");
    }

    return areValidValues;
}

/*
 * Clears all fields displayed in the window.
 */
private void clearDisplay()
{
    lblBookingRef.Content = "Booking number:\r\n";
    lblBookingRef.Visibility = Visibility.Hidden;

    dtpArrival.SelectedDate = null;
    dtpDeparture.SelectedDate = null;

    lblCustNumberValue.Content = String.Empty;
    lblCustNameValue.Content = String.Empty;
    lblCustAddressValue.Content = String.Empty;
}
```

```
/*
 * Refreshes all fields displayed in the window according to
 * current system objects states.
 */
private void refreshDisplay()
{
    clearDisplay();

    refreshBookingDisplay();
    refreshCustomerDisplay();
    refreshGuestsDisplay();
    refreshExtrasDisplay();
}

// METHODS RELATED TO CURRENT BOOKING:

/*
 * Refreshes the booking fields displayed in the window.
 */
private void refreshBookingDisplay()
{
    if (mFacade.IsABookingLoaded())
    {
        DateTime arrival;
        DateTime departure;
        mFacade.GetCurrentBookDates(out arrival, out departure);
        dtpArrival.SelectedDate = arrival;
        dtpDeparture.SelectedDate = departure;

        lblBookingRef.Content += mFacade.GetCurrentBookNb()
                                         .ToString();
        lblBookingRef.Visibility = Visibility.Visible;
    }
}

// METHODS RELATED TO CURRENT CUSTOMER:

/*
 * Refreshes the customer fields displayed in the window.
 */
private void refreshCustomerDisplay()
{
    if (mFacade.IsACustomerLoaded())
    {
        lblCustNumberValue.Content
                              = mFacade.GetCurrentCustNb().ToString();
        lblCustNameValue.Content
                              = mFacade.GetCurrentCustName();
        lblCustAddressValue.Content
                              = mFacade.GetCurrentCustAdress();
    }
}
```

```
/*
 * Loads & displays the customer referenced by txtCustNumber.Text
 */
private void btnCreateLoadCust_Click(object sender, RoutedEventArgs e)
{
    new WindowCustomerDetails(mFacade).ShowDialog();
    refreshCustomerDisplay();
}

// METHODS RELATED TO GUESTS:

/*
 * Refreshes the guests fields displayed in the window.
 */
private void refreshGuestsDisplay()
{
    if (mFacade.IsABookingLoaded())
    {
        lblGuest.Visibility = Visibility.Visible;
        lstGuests.Visibility = Visibility.Visible;
        lstGuests.ItemsSource = mFacade.GetGuestNames();
    }
}

/*
 * Opens an empty WindowGuestDetail dialog to input new guest details
 * (if there is still less than 4 guests booked).
 */
private void btnNewGuest_Click(object sender, RoutedEventArgs e)
{
    if (canAddGuest())
    {
        new WindowGuestDetails(mFacade, false).ShowDialog();
        refreshDisplay();
    }
}

/*
 * True if it is possible to add a guest to the booking,
 * otherwise false.
 * Displays error message windows.
 */
private bool canAddGuest()
{
    bool canAddGuest = true;

    if (!mFacade.IsABookingLoaded())
    {
        canAddGuest = false;
        MessageBox.Show("Please save the booking before adding"
                        + " the guests.");
    }
    else if (mFacade.GetCurrentNbGuests() >= 4)
    {
        canAddGuest = false;
        MessageBox.Show("This booking is already full, the"
                        + " maximum number of guests per booking"
                        + " is 4.");
    }

    return canAddGuest;
}
```

```
/*
 * Opens a WindowGuestDetails dialog to enter guest details for
 * current customer entity.
 */
private void btnAddCustToGuests_Click(object sender, RoutedEventArgs e)
{
    if (canAddGuest())
    {
        if (mFacade.IsGuestACustomer(lstGuests.SelectedIndex))
        {
            MessageBox.Show("The customer is already booked as"
                            + " a guest.");
        }
        else
        {
            new WindowGuestDetails(mFacade, true).ShowDialog();
            refreshGuestsDisplay();
        }
    }
}

/*
 * Opens a WindowGuestDetail to amend selected guest's details.
 */
private void lstGuests_MouseDoubleClick(object sender,
                                        MouseButtonEventArgs e)
{
    try
    {
        List<int> l = (List<int>)lstGuests.ItemsSource;
        int i = lstGuests.SelectedIndex;

        if (mFacade.GetCurrentNbGuests() == 0)
        {
            MessageBox.Show("There is currently no guests booked.");
        }
        else if (i < 0)
        {
            MessageBox.Show("Please double click on the guest that"
                        + " you want to edit");
        }
        else if (mFacade.IsGuestACustomer(i))
        {
            new WindowGuestDetails(mFacade, true, i).ShowDialog();
            refreshGuestsDisplay();
        }
        else
        {
            new WindowGuestDetails(mFacade, false, i).ShowDialog();
            refreshGuestsDisplay();
        }
    }
    catch
    {
        // do nothing
    }

}
```

```
/*
 * Deletes selected guest from current booking.
 */
private void btnDeleteGuest_Click(object sender, RoutedEventArgs e)
{
    if (lstGuests.SelectedIndex < 0 || lstGuests.SelectedIndex > 3)
    {
        MessageBox.Show("Please first select the guest"
                        + " that you want to delete.");
    }
    else
    {
        mFacade.DeleteGuest(lstGuests.SelectedIndex);
        refreshGuestsDisplay();
    }
}

// METHODS RELATED TO EXTRAS:

/*
 * Refreshes the extras fields displayed in the window.
 */
private void refreshExtrasDisplay()
{
    if (mFacade.IsABookingLoaded())
    {
        lstExtras.Items.Clear();
        foreach (String s in mFacade.GetCurrentExtrasNames())
        {
            lstExtras.Items.Add(s);
        }
    }
}

/*
 * Opens a dialog to add a breakfasts extra to the current booking.
 */
private void btnAddBreakfast_Click(object sender, RoutedEventArgs e)
{
    if (!mFacade.IsABookingLoaded())
    {
        MessageBox.Show("Please save your new booking before adding"
                        + " a breakfast extra.");
    }
    else
    {
        new WindowBreakfastDetails(mFacade, null).ShowDialog();
        refreshExtrasDisplay();
    }
}
```

```
/*
 * Opens a dialog to add an evening meals extra to the current booking.
 */
private void btnAddEveningMeal_Click(object sender, RoutedEventArgs e)
{
    if (!mFacade.IsABookingLoaded())
    {
        MessageBox.Show("Please save your new booking before adding"
                        + " an evening meals extra.");
    }
    else
    {
        new WindowEveningMealDetails(mFacade, null).ShowDialog();
        refreshExtrasDisplay();
    }
}

/*
 * Opens a dialog to add a car hire extra to the current booking.
 */
private void btnAddCarHire_Click(object sender, RoutedEventArgs e)
{
    if (!mFacade.IsABookingLoaded())
    {
        MessageBox.Show("Please save your new booking before adding"
                        + " a car hire extra.");
    }
    else
    {
        new WindowCarHireDetails(mFacade, null).ShowDialog();
        refreshExtrasDisplay();
    }
}
```

```
        /*
         * Opens a Window{ExtraType}Details to amend selected extra.
         */
        private void lstExtras_MouseDoubleClick(object sender,
                                                MouseButtonEventArgs e)
        {
            List<BookingDecorator> extras = mFacade.GetCurrentExtras();
            int i = lstExtras.SelectedIndex;

            if (extras != null && extras.Count == 0)
            {
                MessageBox.Show("There is no extra for this booking"
                                + " at present.");
            }
            else if (i < 0)
            {
                MessageBox.Show("Please double click on the extra that"
                            + " you want to edit");
            }
            else
            {
                if (extras.ElementAt(i).GetType() == typeof(Breakfast))
                {
                    new WindowBreakfastDetails(
                            mFacade, extras.ElementAt(i)).ShowDialog();

                }
                else if (extras.ElementAt(i).GetType() == typeof(EveningMeal))
                {
                    new WindowEveningMealDetails(
                            mFacade, extras.ElementAt(i)).ShowDialog();
                }
                else if (extras.ElementAt(i).GetType() == typeof(CarHire))
                {
                    new WindowCarHireDetails(
                            mFacade, extras.ElementAt(i)).ShowDialog();
                }

                refreshExtrasDisplay();
            }
        }

        /*
         * Deletes the selected extra from the currwent booking.
         */
        private void btnExtraDelete_Click(object sender, RoutedEventArgs e)
        {
            if (lstExtras.SelectedIndex >= 0)
            {
                mFacade.RemoveExtra(lstExtras.SelectedIndex);
                refreshExtrasDisplay();
            }
            else
            {
                MessageBox.Show("Please select the extra that you want to"
                                + " delete from the booking");
            }
        }
    }
}
```

30)SourceCode: WindowCustomerDetails.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Program
{
    /*
     * Interaction logic for WindowCustomerDetails.xaml
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-09
     */
    public partial class WindowCustomerDetails : Window
    {
        //PROPERTIES:

        // reference to calling window's ModelFacade instance:
        private ModelFacade mFacade;

        // METHODS:

        /*
         * The window constructor.
         */
        public WindowCustomerDetails(ModelFacade modelFacade)
        {
            this.mFacade = modelFacade;
            InitializeComponent();
            refreshDisplay();
        }

        /*
         * Refreshes all fields displayed in the window according to
         * current system objects states.
         */
        private void refreshDisplay()
        {
            clearDisplay();
            lstCustomers.ItemsSource = mFacade.GetAllCustomerNbs();
            if (mFacade.IsACustomerLoaded())
            {
                lblCustNumberValue.Content = mFacade.GetCurrentCustNb();
                txtCustName.Text = mFacade.GetCurrentCustName();
                txtCustAddress.Text = mFacade.GetCurrentCustAdress();
            }
        }
```

```
/*
 * Refreshes all customer detail fields displayed in the window
 * according to currently loaded customer data.
 */
private void refreshCustDetailDisplay()
{
    if (mFacade.IsACustomerLoaded())
    {
        lblCustNumberValue.Content = mFacade.GetCurrentCustNb();
        txtCustName.Text = mFacade.GetCurrentCustName();
        txtCustAddress.Text = mFacade.GetCurrentCustAdress();
    }
}

/*
 * Empties all boxes displayed in the MainWindow.
 */
private void clearDisplay()
{
    clearCustomerDetails();

    if (lstCustomers.Items != null)
    {
        lstCustomers.Items.Clear();
    }
}

/*
 * Hide empty all customer detail fields from the window.
 */
private void clearCustomerDetails()
{
    lblCustNumberValue.Content = String.Empty;
    txtCustName.Text = String.Empty;
    txtCustAddress.Text = String.Empty;
}

/*
 * loads selected customer into the system when double clicking on it.
 */
private void lstCustomers_MouseDoubleClick(object sender,
                                           MouseButtonEventArgs e)
{
    List<int> l = (List<int>)lstCustomers.ItemsSource;
    int i = lstCustomers.SelectedIndex;

    if (i < 0 || !mFacade.RestoreCustomer(l.ElementAt(i)))
    {
        MessageBox.Show("Please double click on a customer from the"
                    + " list to load it into the system, or create"
                    + " a new one.");
    }

    refreshCustDetailDisplay();
}
```

```
        /*
         * Creates or updates customer, and closes dialog.
         */
        private void btnSave_Click(object sender, RoutedEventArgs e)
        {
            if (areAllValuesValid())
            {
                if (mFacade.IsACustomerLoaded()) // update current customer
                {
                    mFacade.UpdateCurrentCustomer(txtCustName.Text,
                                                  txtCustAddress.Text);
                }
                else // create a new customer
                {
                    mFacade.CreateCustomer(txtCustName.Text,
                                           txtCustAddress.Text);
                }
                this.Close();
            }
        }

        /*
         * True if all fields contain valid values to create/update a
         * customer.
         * Displays error messages.
         */
        private bool areAllValuesValid()
        {
            bool areAllValid = true;

            if (String.IsNullOrWhiteSpace(txtCustName.Text))
            {
                areAllValid = false;
                MessageBox.Show("Please enter a valid customer name");
            }
            else if (String.IsNullOrWhiteSpace(txtCustAddress.Text))
            {
                areAllValid = false;
                MessageBox.Show("Please enter a valid customer address");
            }

            return areAllValid;
        }
    }
}
```

31)SourceCode: WindowGuestsDetails.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Program
{
    /*
     * Interaction logic for WindowGuestDetails.xaml
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-06
     */
    public partial class WindowGuestDetails : Window
    {
        // PROPERTIES:

        // reference to calling window's ModelFacade instance:
        private ModelFacade mFacade;

        // true if the new guest to create is the booking's customer.
        private bool isCustomer;

        // index of current guest in current booking'list of guests.
        private int guestIndex = -1;

        /*
         * Constructor 1: opens a window diplaying no value in any field.
         */
        public WindowGuestDetails(ModelFacade mFacade, bool isCustomer)
        {
            this.mFacade = mFacade;
            this.isCustomer = isCustomer;
            InitializeComponent();

            if (isCustomer)
            {
                hideNameFields();
            }
        }
```

```
/*
 * Constructor 2: opens a window diplaying the detail of element
 * at selected index in booking guests list.
 */
public WindowGuestDetails(ModelFacade mFacade,
                          bool isCustomer,
                          int guestIndex)
{
    this.mFacade = mFacade;
    this.isCustomer = isCustomer;
    this.guestIndex = guestIndex;
    InitializeComponent();

    txtName.Text = mFacade.CurrentBook
                          .GetGuests()
                          .ElementAt(guestIndex)
                          .Name;
    txtPassportNb.Text = mFacade.CurrentBook
                                .GetGuests()
                                .ElementAt(guestIndex)
                                .GetPassportNb();
    txtAge.Text = mFacade.CurrentBook
                         .GetGuests()
                         .ElementAt(guestIndex)
                         .GetAge()
                         .ToString();

    if (isCustomer)
    {
        hideNameFields();
    }
}

/*
 * Hides window fields related to guest name.
 */
private void hideNameFields()
{
    lblName.Visibility = Visibility.Hidden;
    txtName.Visibility = Visibility.Hidden;
}

/*
 * Saves the new guest or guest detail changes.
 */
private void btnSaveGuest_Click(object sender, RoutedEventArgs e)
{
    if (areAllValuesValid())
    {
        if (this.guestIndex >= 0)
        {
            updateGuest();
        }
        else
        {
            createGuest();
        }
        this.Close();
    }
}
```

```
/*
 * True if all the window fields are valid to create a guest,
 * otherwise false.
 * Displays error message windows.
 */
private bool areAllValuesValid()
{
    bool areValidValues = true;
    int tmp;

    if (!Int32.TryParse(txtAge.Text, out tmp))
    {
        MessageBox.Show("Please enter a valid number in age box.");
        areValidValues = false;
    }
    else if (tmp <= 0)
    {
        MessageBox.Show("The guest's age must be strictly greater"
                        + " than zero.");
        areValidValues = false;
    }

    if (!mFacade.IsACustomerLoaded()
     && String.IsNullOrWhiteSpace(txtName.Text))
    {
        MessageBox.Show("Please fill in the guest's name");
        areValidValues = false;
    }

    if (String.IsNullOrWhiteSpace(txtPassportNb.Text))
    {
        MessageBox.Show("Please fill in the guest's passport number");
        areValidValues = false;
    }

    return areValidValues;
}

/*
 * Updates guest details according to window TextBoxes contents.
 */
private void updateGuest()
{
    if (!this.isCustomer) // edit current guest
    {
        mFacade.EditGuest(guestIndex,
                          txtName.Text,
                          txtPassportNb.Text,
                          Int32.Parse(txtAge.Text));

    }
    else // edit current customer's guest properties
    {
        mFacade.EditCustomerGuestDetails(guestIndex,
                                         txtPassportNb.Text,
                                         Int32.Parse(txtAge.Text));
    }
}
```

```
        /*
         * Creates new guest or decorates current customer according to window
         * TextBoxes contents.
         */
        private void createGuest()
        {
            if (!this.isCustomer) // create and add new guest
            {
                mFacade.AddGuest(txtName.Text,
                                 txtPassportNb.Text,
                                 Int32.Parse(txtAge.Text));
            }
            else // decorate current customer as guest and add to list
            {
                mFacade.AddCustomerToGuests(txtPassportNb.Text,
                                            Int32.Parse(txtAge.Text));
            }
        }
    }
}
```

32)SourceCode: WindowBreakfastDetails.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Program
{
    /*
     * Interaction logic for WindowBreakfastDetails.xaml
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-07
     */
    public partial class WindowBreakfastDetails : Window
    {
        //PROPERTIES:

        // reference to calling window's ModelFacade instance:
        private ModelFacade mFacade;

        // points to the Breakfast edited (is null when creating new one).
        private BookingDecorator breakfast;
```

```
        // METHODS:

        /*
         * Constructor, the index passed must be the index in the current
         * booking's decoration stack as returned by
         * ModelFacade.GetCurrentExtras()) or -1 for a new extra.
         */
        public WindowBreakfastDetails(ModelFacade mFacade,
                                      BookingDecorator instance)
        {
            this.mFacade = mFacade;
            this.breakfast = instance;
            InitializeComponent();

            if (this.breakfast != null)
            {
                txtDietRequirements.Text
                            = ((Breakfast)instance).GetDietRequirements();
            }
            else
            {
                txtDietRequirements.Text = String.Empty;
            }
        }

        /*
         * Creates or updates the extra.
         */
        private void btnSave_Click(object sender, RoutedEventArgs e)
        {
            if (this.breakfast != null)
            {
                mFacade.UpdateBreakfast(this.breakfast,
                                        txtDietRequirements.Text);
            }
            else
            {
                mFacade.AddBreakFast(txtDietRequirements.Text);
            }

            this.Close();
        }
    }
}
```

33)SourceCode: WindowEveningMealDetails.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Program
{
    /*
     * Interaction logic for WindowEveningMealDetails.xaml
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-08
     */
    public partial class WindowEveningMealDetails : Window
    {
        //PROPERTIES:

        // reference to calling window's ModelFacade instance:
        private ModelFacade mFacade;

        // points to the EveningMeal edited (is null when creating new one).
        private BookingDecorator eveningMeal;

        // METHODS:

        /*
         * Constructor, the index passed must be the index in the current
         * booking's decoration stack index as returned by
         * ModelFacade.GetCurrentExtras()) or -1 for a new extra.
         */
        public WindowEveningMealDetails(ModelFacade mFacade,
                                        BookingDecorator instance)
        {
            this.mFacade = mFacade;
            this.eveningMeal = instance;
            InitializeComponent();

            if (this.eveningMeal != null)
            {
                txtDietRequirements.Text
                            = ((EveningMeal)instance).GetDietRequirements();
            }
            else
            {
                txtDietRequirements.Text = String.Empty;
            }
        }
```

```
        /*
         * Creates or updates the extra.
         */
        private void btnSave_Click(object sender, RoutedEventArgs e)
        {
            if (this.eveningMeal != null)
            {
                mFacade.UpdateEveningMeal(this.eveningMeal,
                                          txtDietRequirements.Text);
            }
            else
            {
                mFacade.AddEveningMeal(txtDietRequirements.Text);
            }

            this.Close();
        }
    }
}
```

34)SourceCode: WindowCarHireDetails.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Program
{
    /*
     * Interaction logic for WindowCarHireDetails.xaml
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-08
     */
    public partial class WindowCarHireDetails : Window
    {
        //PROPERTIES:

        // reference to calling window's ModelFacade instance:
        private ModelFacade mFacade;

        // points to the CarHire extra edited (is null when creating new one).
        private BookingDecorator carHire;
```

```
// METHODS:

/*
 * Constructor, the index passed must be the index in the current
 * booking's decoration stack index as returned by
 * ModelFacade.GetCurrentExtras()) or -1 for a new extra.
 */
public WindowCarHireDetails(ModelFacade mFacade,
                            BookingDecorator instance)
{
    this.mFacade = mFacade;
    this.carHire = instance;
    InitializeComponent();

    if (this.carHire != null)
    {
        DateTime start;
        DateTime end;
        ((CarHire)instance).GetHireDates(out start, out end);
        dtpStart.SelectedDate = start;
        dtpEnd.SelectedDate = end;

        txtDriverName.Text = ((CarHire)instance).GetDriverName();
    }
    else
    {
        txtDriverName.Text = String.Empty;
    }
}

/*
 * Creates or updates the extra.
 */
private void btnSave_Click(object sender, RoutedEventArgs e)
{
    if (areAllValuesValid())
    {
        if (this.carHire != null)
        {
            mFacade.UpdateCarHire(this.carHire,
                                  txtDriverName.Text,
                                  (DateTime)dtpStart.SelectedDate,
                                  (DateTime)dtpEnd.SelectedDate);
        }
        else
        {
            mFacade.AddCarHire(txtDriverName.Text,
                               (DateTime)dtpStart.SelectedDate,
                               (DateTime)dtpEnd.SelectedDate);
        }
        this.Close();
    }
}
```

```
        /*
         * True if all the window fields are valid to create a car hire extra,
         * otherwise false.
         * Displays error message windows.
         */
        private bool areAllValuesValid()
        {
            bool areValidValues = true;
            DateTime arrival;
            DateTime departure;
            mFacade.GetCurrentBookDates(out arrival, out departure);

            if (String.IsNullOrWhiteSpace(txtDriverName.Text))
            {
                areValidValues = false;
                MessageBox.Show("Please enter a driver name for the"
                                + "car hire.");
            }
            else if (dtpStart.SelectedDate == null)
            {
                areValidValues = false;
                MessageBox.Show("Please select a start date for the"
                                + " car hire.");
            }
            else if (dtpStart.SelectedDate < arrival
                  || dtpStart.SelectedDate >= departure)
            {
                areValidValues = false;
                MessageBox.Show("The selected start date is outwith "
                                + " the booking dates.\r\n"
                                + "Please select a start date between"
                                + " booking arrival and departure dates.");
            }
            else if (dtpEnd.SelectedDate == null)
            {
                areValidValues = false;
                MessageBox.Show("Please select an end date for the"
                                + " car hire.");
            }
            else if (dtpEnd.SelectedDate <= arrival
                  || dtpEnd.SelectedDate > departure)
            {
                areValidValues = false;
                MessageBox.Show("The selected end date is outwith "
                                + " the booking dates.\r\n"
                                + "Please select a start date between"
                                + " booking arrival and departure dates.");
            }

            return areValidValues;
        }
    }
}
```

35)SourceCode: WindowInvoice.xaml.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace Program
{
    /*
     * Interaction logic for MainWindow.xaml
     *
     * author: Pierre Ruiz (matriculation number 08009004)
     * last modified: 2016-12-07
     */
    public partial class WindowInvoice : Window
    {
        // PROPERTIES:

        // reference to a ModelFacade instance:
        private ModelFacade mFacade;


        // METHODS:

        /*
         * Constructor.
         */
        public WindowInvoice(ModelFacade mFacade)
        {
            InitializeComponent();
            this.mFacade = mFacade;

            // get all the amounts:
            int nbNights = mFacade.GetCurrentNbNights();
            float costPerNight = mFacade.GetCurrentCostPerNight();
            float breakfastCost = mFacade.GetCurrentBreakfastCost();
            float eveningMealsCost = mFacade.GetCurrentEveningMealsCost();
            float carHireCost = mFacade.GetCurrentCarHireCost();

            printDetails();
            printBookingBreakdown(nbNights, costPerNight);
            printExtrasBreakdown(breakfastCost, eveningMealsCost, carHireCost);
            printTotal((nbNights * costPerNight)
                            + breakfastCost
                            + eveningMealsCost
                            + carHireCost);
        }
```

```
        /*
         * Fills the labels of the invoice concerning the booking
         * details.
         */
        private void printDetails()
        {
            lblBookingNb.Content
                += " " + mFacade.GetCurrentBookNb().ToString();

            lblCustomerNb.Content
                += " " + mFacade.GetCurrentCustNb().ToString();

            lblCustomerDetails.Content
                += " " + mFacade.GetCurrentCustName();

            lblAddress.Content
                = mFacade.GetCurrentCustAdress();
        }

        /*
         * Fills the labels of the invoice concerning the booking
         * cost.
         */
        private void printBookingBreakdown(int nbNights, float costPerNight)
        {
            lblNbNightsValue.Content = nbNights;
            lblCostPerNightValue.Content = costPerNight;
            lblSubtotalValue.Content = (costPerNight * nbNights);
        }

        /*
         * Fills the labels of the invoice concerning the extras
         * costs.
         */
        private void printExtrasBreakdown(float breakfastCost,
                                          float eveningMealsCost,
                                          float carHireCost)
        {


            lblBreakfastCostValue.Content = breakfastCost;
            lblEveningMealsCostValue.Content = eveningMealsCost;
            lblCarHireCostValue.Content = carHireCost;

            lblExtrasTotalValue.Content = (breakfastCost
                                           + eveningMealsCost
                                           + carHireCost);
        }

        /*
         * Fills the labels of the invoice concerning the booking
         * total due.
         */
        private void printTotal(float grandTotal)
        {
            lblTotalDueValue.Content = grandTotal;
        }
    }
}
```

36) Source code: UnitTest1.cs

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace Program
{
    [TestClass]
    public class UnitTest1
    {
        [TestMethod]
        public void TestBookingConstrutor_NormalCase()
        {
            // arrange
            PersonComponent c = PersonFactory.Instance
                                             .GetNewCustomer("name","address");

            BookingComponent b1 = new Booking(1,
                                              c,
                                              new DateTime(2016, 12, 09),
                                              new DateTime(2016, 12, 13));

            BookingComponent b2 = new Booking(1,
                                              c,
                                              new DateTime(2016, 12, 09),
                                              new DateTime(2016, 12, 13));

            // assert
            Assert.IsTrue(b1.Equals(b2), "Booking.Equals method problem");
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentException))]
        public void TestBookingConstructor_CustomerException()
        {
            // arrange
            PersonComponent p = new Person("name");

            // act
            BookingComponent b = new Booking(98,
                                             p,
                                             new DateTime(2016, 02, 18),
                                             new DateTime(2016, 03, 2));
        }

        [TestMethod]
        [ExpectedException(typeof(ArgumentException))]
        public void TestBookingConstructor_BookingNbException()
        {
            // arrange
            PersonComponent c = PersonFactory.Instance
                                             .GetNewCustomer("name", "address");
            // act
            BookingComponent b = new Booking(-10,
                                             c,
                                             new DateTime(1998, 02, 18),
                                             new DateTime(1998, 03, 2));
        }
```

```
[TestMethod]
[ExpectedException(typeof(ArgumentException))]
public void TestBookingConstructor_DatesException()
{
    // arrange
    PersonComponent c = PersonFactory.Instance
                                     .GetNewCustomer("name", "address");
    // act
    BookingComponent b = new Booking(3,
                                     c,
                                     new DateTime(2016, 04, 18),
                                     new DateTime(2016, 02, 15));
}

[TestMethod]
public void TestBookingAddGuest_NormalCase()
{
    // arrange
    PersonComponent expected = PersonFactory.Instance
                                            .GetNewGuest("myself",
                                                         "08855N8",
                                                         31);
    BookingComponent b1 = new Booking(
                                 1,
                                 PersonFactory.Instance
                                              .GetNewCustomer("name",
                                                              "address"),
                                 new DateTime(1024, 12, 09),
                                 new DateTime(2016, 9, 3));
    // act
    b1.AddGuest(expected);

    // assert
    Assert.IsTrue(b1.GetGuests().Contains(expected),
                  "Problem with Booking.AddGuest");
}

[TestMethod]
public void TestBookingGetNbGuests_NormalCase()
{
    // arrange
    int expected = 0;
    BookingComponent b1 = new Booking(
                                 1,
                                 PersonFactory.Instance
                                              .GetNewCustomer("name",
                                                              "address"),
                                 new DateTime(1024, 12, 09),
                                 new DateTime(2016, 9, 3));
    // act
    int result = b1.GetNbGuests();

    // assert
    Assert.AreEqual(expected, result,
                    "Problem with Booking.GetNbGuests");
}
```

```
        [TestMethod]
        public void TestBookingGetNbNights_NormalCase()
        {
            // arrange
            int expected = 4;
            BookingComponent b1 = new Booking(
                                        1,
                                        PersonFactory.Instance
                                                     .GetNewCustomer("name",
                                                                     "address"),
                                        new DateTime(2010, 01, 07),
                                        new DateTime(2010, 01, 11));
            // act
            int result = b1.GetNbNights();

            // assert
            Assert.AreEqual(expected, result,
                            "Problem with Booking.GetNbNights");
        }

        [TestMethod]
        public void TestBookingGetCostPerNight_NormalCase()
        {
            // arrange
            float expected = 80;
            BookingComponent b1 = new Booking(
                                        1,
                                        PersonFactory.Instance
                                                     .GetNewCustomer("name",
                                                                     "address"),
                                        new DateTime(2010, 01, 07),
                                        new DateTime(2010, 01, 11));
            b1.AddGuest(PersonFactory.Instance.GetNewGuest("g1", "Pass", 57));
            b1.AddGuest(PersonFactory.Instance.GetNewGuest("g2", "Port", 14));

            // act
            float result = b1.GetCostPerNight();

            // assert
            Assert.AreEqual(expected, result,
                            "Problem with Booking.GetCostPerNight");
        }
    }
}
```

CLASS DIAGRAM
(please open the project uploaded on moodle into visual studio for better definition)