CRC

CRC = Class, Responsibilities, Collaborations

**Structure of a CRC card:** on a small card, record:

1. at the top: the name of the class
2. on the left-hand side: the responsibilities of the class
3. on the right-hand side: the collaborators of the class, which help to carry out each responsibility

**Tips:**

1. Responsibilities should describe at a high level the purpose of the existence of the class.
2. A class should normally have no more than 3-4 responsibilities. If more, consider rephrasing the responsibilities more concisely, or breaking the class in two.
3. Too many responsibilities indicates low cohesion in the model; too many collaborators indicates high coupling. Both should be avoided.
CRC cards: example

<table>
<thead>
<tr>
<th>Client</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Provide financial information</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cottage</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Keep track of current occupation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keep track of future reservations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Print out past occupation and cleaning needs</td>
<td>List, Summary</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Meeting room</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Keep track of current occupation</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Keep track of future reservations</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Print out past occupation and cleaning needs</td>
<td>List, Summary</td>
</tr>
</tbody>
</table>
CRC cards (cont’d)

Suggested methodology:

1. Walk through use cases; work out how the class model provides the functionality required by the use cases, and where the missing bits are.
2. Role playing can be used when working in a team
3. If there is a missing link (something has to be done, but no class has the responsibility of doing it) it means the design is faulty or incomplete.
CRC cards (cont’d)

Use case (without alternate paths): a person makes a reservation for a cottage
1. The person calls in and requests a reservation for a cottage with a certain number of beds for a certain date.
2. The system checks for availability of suitable cottages and selects a cottage.
3. The system creates a new client record.
4. The system creates a reservation for the client, for the selected cottage.
This case points out to the possibility that the Reservation object should be moved to the list of 'good' classes.
Discussion!!!
Class diagram

Simple: a box with the name of the class

Developed: with attributes and operations

<table>
<thead>
<tr>
<th>Circle</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPos: double</td>
</tr>
<tr>
<td>YPos: double</td>
</tr>
<tr>
<td>Radius: double</td>
</tr>
<tr>
<td>Color: COLOR</td>
</tr>
<tr>
<td>Translate(x: float, y: float)</td>
</tr>
<tr>
<td>IncludesOrigin(): boolean</td>
</tr>
</tbody>
</table>

Note: you can include visibility attributes: + for public, - for private, # for protected
Association
"Car Simulation System requirements statement"

- The system shall store car information such as gasQuantity.
- A user shall be able to:
  - start the car
  - operate the car
  - stop the car
  - add a passenger
  - remove a passenger
  - make a cellular phone call
  - receive a cellular phone call.
External Car User

Car Simulation System

- Car
- Tire
- Tire
- Motor
- Cellular Phone
- Passenger

(has a)
(part of)
(has many)
generalization specialization (Vehicle to Car)
1 to 1 aggregation (Car to Motor)
1 to many aggregation (Car to Tire)
1 to zero or one association (Car to CellularPhone)
1 to many association (Car to Passenger)
CellularPhone *currentCellularPhone;

Car () { currentCellularPhone = 0; } // default constructor to initialize currentCellularPhone pointer
Car (CellularPhone *aCellularPhone) //constructor with argument to connect the associated object
{ currentCellularPhone = aCellularPhone; }

~Car () { currentCellularPhone = 0; } // destructor to detach associated object

The following are sample invocations of these two constructors:
Car car1; // invokes the default constructor
Car car2 (&cellularPhone1) // invokes the constructor with arguments
```cpp
#ifndef __CAR_H
#define __CAR_H

#ifndef __CLLLPRPHN_H
#endif

#include "ClllrPhn.h"
#endif

class Car {
    int speed; // Attribute data member
    CellularPhone* currentCellularPhone; // 1:1 association object data member

public:
    Car() : speed(0) { currentCellularPhone = 0; // Initialization to 0 }

    // Get accessor function for 1:1 association object data member
    const CellularPhone* getCellularPhone() const
    { return currentCellularPhone; }

    // Set accessor function for 1:1 association object data member
    void setCellularPhone(CellularPhone* const acurrentCellularPhone)
    { currentCellularPhone = acurrentCellularPhone; }

    // Remove function for 1:1 association object data member
    // Warning delete currentCellularPhone object if dynamic object
    void removeCellularPhone()
    { currentCellularPhone = 0; }

    void makePhoneCall(long aNumber);

    ~Car(); // Destructor

};
#endif
```
#ifndef __CLLRPHN_H
#define __CLLRPHN_H

class CellularPhone
{
    long phoneNumber; //Attribute data member

    public:
    CellularPhone () : phoneNumber(0) { }
    void makePhoneCall (long aNumber) ;

    ~ CellularPhone () { } //Destructor

};
#endif
#include "Car.h"

void Car::makePhoneCall(long aNumber) //Body added after code generation
{
    if (currentCellularPhone == 0) return; //no cellular phone to make call
    currentCellularPhone -> makePhoneCall (aNumber);
}

#include "ClllrPhn.h"

void CellularPhone::makePhoneCall(long aNumber) {
}

Car
speed
makePhoneCall
setCellularPhone
getCellularPhone
removeCellularPhone

CellularPhone
phoneNumber
makePhoneCall

currentCellularPhone
```cpp
#include "Car.h"

int main ()
{
    Car car1; //invokes the default constructor
    CellularPhone cellularPhone1;
    car1.setCellularPhone (&cellularPhone1);
    car1.makePhoneCall (55555);
    return 0;
}
```
Resort reservation: class interaction

- **Client**
  - 1 association with **Reservation**

- **Reservation**
  - N association with **Bookable Item**

- **Bookable Item**
  - 1 association with **Report**
  - 1 association with **Cottage**
  - 1 association with **Meeting Room**
Implementation

The implementation that follows assume a type TIME defined for example as:

```c
struct TIME
{
    int Year;
    int Month;
    int Day;
    int Hour;
    int Minute;
};
```
class Client
{
    static const int MAX_NAME = 80;
    static const int MAX_PHONE = 50;
    enum CCTYPE {VISA, MASTERCARD, AMEX};

    class Client
    |
    // Data members
    protected:
        char itsName[MAX_NAME];
        char itsPhoneNumber[MAX_PHONE];
        CCTYPE itsCCTYPE;
        long itsCCNumber;
        TIME itsCCExpiryDate;

    // Member functions
    public:
        const char *GetName();
        void SetName(const char* Name);
        const char *GetPhoneNumber();
        void SetPhoneNumber(const char* PhoneNumber);
        void GetCCInfo(CCTYPE &Type, long &Number, TIME &ExpiryDate);
        void SetCCInfo(CCTYPE Type, long Number, TIME ExpiryDate);
};
class Reservation
{
    // Data members
    protected:
    Client *itsClient; // pointer to client
    TIME itsStart;
    TIME itsEnd;
    int itIsMoveable;
    int itsTrackingNumber;

    // Definition of reservation as a linked list
    Reservation *itsNext;

    // Member functions
    public:
    int IsMoveable();
    TIME GetStart();
    // etc..
};
class BookableItem
{
    // Data members
    protected:
        Reservation *itsResList; // Reservations are in a linked list

    // Member functions
    public:
        virtual int Book(TIME Start, TIME End) = 0; // pure virtual
        virtual void PrintOccupation(Report &Doc);
        virtual void PrintCleaningNeeds(Report &Doc);
};
Books and Copies

```
Book
- title : String
- author : String
- desc : String
- pages : int
```

```plaintext
BookCopy
- copyNum : int
```
class BookCopy;

class Book{
public:
    // ...
    BookCopy* addCopy();
    BookCopy* getAvailable() const;
protected:
private:
    // ...
    List copies_; // <BookCopy>
};

class Book{
public:
    // ...
    BookCopy* addCopy();
    BookCopy* getAvailable() const;
protected:
private:
    // ...
    List copies_; // <BookCopy>
};

class BookCopy{
public:
    // ...
    BookCopy(Book* b, int copyNum);
    int checkoutLength() const;
    int isCheckedOut() const;
    int copyNum() const;
private:
    Book* book;
    int copyNum;
    // CheckoutItem* checkout_; // wait...
};
Books and Copies

Diagram:

- Book
  - pages_int=505
  - author_String= "Peter Coad"
  - title_String=
  - desc_String=

- BookCopy
  - copyNum_int=1
  - book_Book*

- BookCopy
  - copyNum_int=2
  - book_Book*
Client and Consultant
class Client {
    ...
    private:
        string company;
        list<Contract*> contracts;
};

class Consultant {
    ...
    private:
        string name;
        list<Contract*> contracts;
};

class Contract {
    ...
    private:
        // Associated links
        Client* client;
        Consultant* consultant;
        // Regular attributes
        double rate;
    ...
};