Exercise Sheet 6

Exercise 1
There is stamp coupling between two software units if one of these units passes complex data structures to the other unit (e.g., as parameters in a method call). Suggest a way to refine such a design to avoid stamp coupling.

Solution:

There are two ways to reduce stamp coupling,

1. Passing simple variables
2. Using interface as the argument type

1. Using Simple Variables to avoid stamp coupling:

```
Public class Emailer {
        Public void senEmail (Employee e, String text)
        {…}
….
}
```

Replace "Employee" data type with simple data types.

```
Public class Emailer {
        Public void senEmail (String name, String email, String text)
        {…}
….
}
```

2. Using Interface to avoid it:

```
Public interface Addressee{
        Public abstract String getName ();
        Public abstract String getEmail ();
….
}

Public class Employee implements Addressee {….}

Public class Emailer {
        Public void senEmail (Addressee e, String text)
        {…}
….
}
```

Exercise 2
Exercise 2.1
Are there systems that cannot be made completely functionally cohesive? Justify your decision.

Solution:
Functional cohesion is achieved when a module only performs a single computation, and returns a result, without having side-effects. While functional cohesion is considered the most desirable type of cohesion for a software module, but it may not be achievable.
Calculating Employee net salary could be an example which cannot be functionally cohesive because it's performing one job however we knows that it's computing gross salary, FICA deductions, federal income tax, state income tax, and so on. Here every sub module could be functionally cohesive but the system is not.

Exercise 2.2
Can a system ever be completely decoupled? That is, can the degree of coupling be reduced so much that there is no coupling between components? Justify your decision.

Solution:
A system cannot be completely decoupled, because if the components are completely decoupled they become independent system them self.
The least coupling between system components is at the level of coincidental cohesion, where the parts have no meaningful relationship. Even then they are not completely independent.

Exercise 3
Exercise 3.1
For each type of cohesion, write a description of a component exhibiting that kind of cohesion.

Solution:

Coincidental Cohesion:
A coincidentally cohesive module is one whose elements contribute to activities with no meaningful relationship to one another
In the following example findPattern, average and openFile are not related with each others.

```
Public class Rous{

  public static int findPattern( String text, String pattern)
    { … }
  public static int average( Vector numbers )
    { … }
  public static OutputStream openFile( String fileName )
    { … }
}
```

Logical Cohesion:

A logically cohesive module contains a number of activities of the same general kind. To use the module, we pick out just the pieces we need. These parts are related by the logical structure of code. Codes proceeded by if-then-else or switch statements.

Following is an example of Logical cohesion. Where a parameter 'flag' is passed to a function 'sample'. And on the bases of value (ON, OFF, CLOSE), the code will be executed.

```
public void sample( String flag ) {
        switch ( flag ){
                case ON:
                        …….
                        break;
                case OFF:
                        ……
                        break;
                case CLOSE:
                        …….
                        break;
        }
}
```

Temporal Cohesion:

A temporally cohesive module is one whose elements are involved in activities that are related in time or we can say elements are grouped into a module because they are all processed within the same limited time period for example "Initialization" modules that provide default values for objects "End of Job" modules that clean up.

Following is a simple example of initializing variables in constructor and then destroying them in destructor.

```
Public class foo {
        Private string name;
        Private int size;
                //constructor
        public void foo(){

                this.name = "Not Set";
                this.size = 12;
        }
                //destructor
        Public void ~foo() {
                delete[] name;
                delete size;
        }
  }
```

Procedural Cohesion:

A procedurally cohesive module is one whose elements are involved in different and possibly unrelated activities in which control flows from each activity to the next.

Following is an example of procedural Cohesion, where at the end of month a procedure "MonthEnd" is calculating expenses, revenue and employee reports by calling other functions and then printing them.

```
void MonthEnd()
{
  Report ExR = InitExpenseReport();
  Report rr = InitRevenueReport();
  Report EmpR = InitEmployeeReport();

  EmpR.Init();
  rr.Init();
  ExR.SetEmployees(true);

  if (ExR.GetReportParams())
    EmpR.GetReportParams();

  SendToPrinter(rr);
  SendToPrinter(ExR);
  SendToPrinter(EmpR);
}
```

Communication Cohesion:

A module has communicational cohesion if it performs a series of actions related by the procedure to be followed by the product, but in addition all the actions operate on the same data

For example calculating a product of two numbers and saving it in database.

```
Public class Calculate {
        Public int product;
        public void product(int a, int b){
                product = a*b;
                ….
                save(product);
                ….
        }
        Public void save(int product) {
                \\ code to store value into database
                …
        }
  }
```

Functional Cohesion:
If the operations of a module can be collectively described as a single specific function in a coherent way, the module has functional cohesion

Examples for this could be, copying file to the floppy, calculating sales commission.

```
public int commission(int sale, long percentage){
        int com;
        //calculate commission
        return com;
    }
```

Informational Cohesion:

A module has informational cohesion if it performs a number of actions, each with its own entry point, with independent code for each action, all performed on the same data structure. This cohesion is supported by data abstraction and Object Oriented design.

```
Public interface Addressee{
    ……
    Public abstract String getName ();
    Public abstract String getAddress ();
    ….
}

Public class Employee implements Addressee {….}
```

Exercise 3.2
For each type of coupling, give an example of two components coupled in that way.

Solution:

Content coupling:
Occurs when one component surreptitiously modify data that is internal to another component.

```
public class Vector3D{
    public int x, y, z;
    ...
}
public class Arch {
    private Vector3D baseline;
    ...
    void slant(int newY){
        baseline.x = 10;
        baseline.y = 13;
    }
}
```

Control Coupling:

Occurs when one procedure calls another using a ‘flag’ or ‘command’ that explicitly controls what the second procedure does.

```
public routineX(String command) {
        if (command.equals("drawCircle"){
          drawCircle();
        }
        else{
          drawRectangle();
        }
}
```

Stamp Coupling:

Occurs whenever one of your application classes is declared as the *type* of a method argument. Mean when a complex data structure is passed as argument in a function call.

```
public class Employee{
        public String name, emailID.
  ...
}


public class Emailer{
        public void sendEmail(Employee e, String text)
        {...}
  ...
}
```

Data coupling:

Data coupling is desirable in some specific systems where the Output from one module is the input to another.

```
Public class Receiver {

   public void message( MyType X ){

                        …
                        X.doSomethingForMe( Object data );
                        …
     }
   }
```

Common Coupling:

When two modules refer to the same global data area, they are common coupled. There are some systems such as customer billing system so that modules can only perform tasks through the global variables.

In the following example global variable **x** is shared with two functions **addValue** and **subtractValue**:

```
int x;

public class myValue{

        public void addValue(int a){
                x = x+ a;
        }
        public void  subtractValue(int a){
                x = x- a;
        }

}
```