BAB II
LANDASAN TEORI

II.1 Electronic Ticketing

Tiket Elektronik (*Electronic Ticketing*) merupakan teknologi yang digunakan dalam pemesanan tiket secara online. Tiket elektronik telah dikembangkan untuk mengganti tiket kertas dan *voucher* kedalam bentuk elektronik. Keuntungan digunakannya tiket elektronik adalah pengguna mampu membeli tiket masuk, *voucher* makan dan minuman yang didownload pada suatu kartu. Semua orang dapat menggunakan smartphone untuk memesan tiket bukan menunggu dalam antrian untuk membeli kertas tiket. Tiket elektronik digunakan untuk mempercepat transaksi dan memudahkan penggunanya dalam pemesanan tiket [7]. Penjualan tiket secara online banyak diterapkan termasuk penjualan tiket suatu event.

II.2 Object Oriented Programming

Object Oriented Programming merupakan sebuah pendekatan pengembangan perangkat lunak, dimana struktur perangkat lunak dianalisis berdasarkan objek yang saling berinteraksi untuk menyelesaikan tugasnya berdasarkan urutan logika yang telah ditentukan. Interaksi ini membutuhkan pertukaran pesan bolak-balik antara objek yang satu dengan objek yang lainnya. Setelah pesan diterma, objek dapat melakukan action atau method [8].

Object Oriented programming memiliki enam konsep dasar, yaitu:

1. Object

   Object merupakan struktur yang menggabungkan dara dan prosedur agar dapat berjalan dengan suatu data.

2. Abstraction

   Abstraction merupakan kemampuan untuk menyaring sejumlah informasi dengan kata lain tidak perlu untuk mengetahui seluruh sifat-sifat asing suatu objek tetapi hanya menggunakan apa yang diperlukan saja. Kemampuan abstaksi diperlukan karena sangat sulit untuk memproses sejumlah informasi tanpa kemampuan untuk menyaring sifat objek.

3. Encapsulation

   Enkapsulation adalah proses dimana akses data secara langsung tidak diizinkan. Untuk mengakses data tersebut dapat menggunakan objek yang bertanggung jawab dalam pengaksesan data. Dengan enkapsulasi data membuat sistem yang dibuat lebih aman.

4. Inheritance

   Inheritance atau pewarisan digunakan untuk mengklasifikasikan objek menuruk karakteristik dan fungsi. Dengan menggunakan inheritance membuat pekerjaan dengan menggunakan objek menjadi lebih mudah.

5. Polymorphysm

   Polymorphysm adalah kemampuan dua bauh objek berbeda untuk menerima pesan permintaan yang sama dengan cara yang berbeda pada tiap objek. Pada implementasinya dapat menggunakan objek dengan nama yang sama.

6. Agregation

   Agregation adalah dimana ketika sebuah objek yang terdiri dari gabungan objek yang saling bekerjasama. Dengan menggunakan agregasi memungkinkan untuk mengimplementasitakan prosen bisnis sesusai dengan model.

II.3 Style Programming C#

Setiap bahasa pemrograman pasti memiliki gaya sendiri dalam penulisan programnya, begitu juga pada bahasa pemrograman C#. *Style programming* memiliki tujuan untuk menciptakan penulisan kode yang konsisten sehingga orang yang membacanya hanya perlu berfokus pada konten dan memungkinkan untuk memahami kode dengan cepat.

1. Penamaan

Penggunaan nama-nama yang bermakna artinya nama variabel harus benar-benar menjelaskan fungsinya. Nama variabel harus kata atau kombinasi kata kecuali bila suatu varibel berupa huruf dikenali, seperti i dan j. Penamaan menggunakan *Pascal Case* pada setiap awal kata menggunakan huruf kapital sedangkan *Camel Case* sama dengan *pascal case* hanya huruf pertama pada kata pertama tidak dengan huruf kapital [9].

Identifier	Case	Example
Class	Pascal	*AppDomain*
Enumeration type	Pascal	*ErrorLevel*
Enumeration values	Pascal	*FatalError*
Event	Pascal	*ValueChanged*
Exception	Pascal	*WebException*
Read-only static field	Pascal	*RedValue*
Interface	Pascal	*IDisposable*
Method	Pascal	*ToString*
Namespace	Pascal	*System.Drawing*
Parameters and Locals	Camel	*typeName*
Property	Pascal	*BackColor*

Gambar II.2 Penamaan pada bahasa pemrograman C#

```
var currentPerformanceCounterCategory = new System.Diagnostics.
    PerformanceCounterCategory();
```

Gambar II.3 Contoh penulisan nama

Nama kelas interface diawali dengan "I" contohnya adalah *IDisposable*, sedangkan *exception* pasti diakhiri dengan "Exception" contohnya *WebException*. Utilize Namespace setiap kelas harus ditutup dalam beberapa namespace. Berikut adalah contoh penulisan namespace.

```
namespace Homework1
{
    class MySolution
    {
        /* ... */
    }
}
```

Gambar II.4 Penulisan namespace

2. Tata Letak (*Layout*)

   Layout yang baik menggunakan format untuk menandai struktur kode sehingga lebih mudah untuk dibaca. Gunakan pengaturan Code Editor yang default. Penulisan statement hanya boleh satu statement per baris. Deklarasi hanya boleh satu deklarasi per baris. Apabila baris yang berkelanjutan tidak terindentasi secara otomatis maka gunakan satu kali *tab stop*. Tambah sebuah baris kosong antara definisi method dan definisi property. Gunakan tanda kurung untuk membuat klausa dalam ekspersi yang jelas. Berikut adalah contoh *layout* yang benar [10].

```
if ((val1 > val2) && (val1 > val3))
{
    // Take appropriate action.
}
```

Gambar II.5 Contoh penulisan layout

3. *Comment*

   Penulisan *comment* yang baik adalah dengan menempatkan di sebuah baris terpisah jangan ditempatkan di akhir kode. Awali *comment* dengan huruf besar. Tambahkan satu spasi antara pembatas *comment* (//) dan teks *comment*. Berikut ini adalah contoh penulisan comment [10].

```
// The following declaration creates a query. It does not run
// the query.
```

Gambar II.6 Contoh penulisan *comment*

4. Indentasi dan Spasi

Maksimal kolom yang digunakan adalah 80 kolom artinya maksimal penggunaan karakter adalah 80 karakter per baris. Apabila menggunakan lebih dari 80 kolom menyebabkan kode di teruskan ke baris selanjutnya. Hal ini dapat menyulitkan saat kode dibaca [9].

Jangan menggunakan krakter tab. Untuk ngganti tab dapat menggunakan spasi untuk mengontrol indentasi. Kebanyakan baris pada kode yang diindent hanya boleh empat spasi dari baris kode sebelumnya. Baris kosong hanya boleh ada antara deklarasi dan harus dibatasi maksimal satu baris kosong [9]. Contoh penggunaan indentasi dan spasi adalah sebagai berikut.

```
public bool LargeMethod(FirstVeryLargeParamterType t1,
                        SecondVeryLargeParameterType t2,
                        ShortThirdParameterType t3)
{
    return (SomeBooleanFunction(t1, t2)    ||
            AnotherBooleanFunction(t2, t3) ||
            SomeBooleanFunction(t1, t3));
}
```

Gambar II.7 Contoh penggunaan indentasi dan spasi

II.4 Generic Programing

Generic Programming bertujuan untuk meringankan beban programmer dengan cara memungkinkan fungsi umum untuk di definisikan sekali untuk banyak tipe data, bukan hanya untuk satu tipe data. Generic programming menggunakan menambahkan parameter program generik.

II.5 Class Library

Class Library merupakan sebuah *library* (perpustakaan) yang berisi kumpulan kelas-kelas, *interface*, dan tipe data. *Class library* memiliki fungsi untuk menyediakan akses ke fungsionalitas sistem dan dirancang untuk menjadi fondasi dari aplikasi, komponen-komponen dan kontrol yang dibangun [11]. Class library akan di simpan dalam sebuah file dengan extensi .dll dan kemudian baru dapat digunakan. Penamaan pada class libarary dapat menggunakan *pascal case* yaitu

huruf pertama pada setiap kata dalam penamaan selalu diawali dengan huruf besar. Dengan menggunakan *pascal case* nama dari identifier akan lebih mudah untuk dibaca [12].

II.5.1 Pedoman Pembangunan Class Library

Pedoman pembangunan Class Library ini merupakan pedoman yang digunakan pada .NET Framework. Tujuan dari pedoman ini adalah untuk membantu desainer class library untuk memstikan bahwa class library yang dibangunnya paham karena konsistensi API dan kemudahan penggunaan dengan menyediakan model pemrograman yang terpadau dan independen. Desain class library yang tidak konsisten akan merugikan produktifitas pengembang [13]. Berikut adalah pedoman yang harus diperhatikan dalam membangun class library.

1. *Naming Guidelines*

   Penamaan dalam pembangunan Class Library merupakan hal yang penting karena hal ini diperlukan agar mudah dipahami oleh pengguna. Terdapat beberapa aturan penamaan yaitu *General Naming Conventions, Names of Namespaces, Name of Clases, Stucts, and Interfaces, Name of Type Members, Names of Parameters, Names of Resources*. Pola penamaan yang konsisten akan mempengaruhi prediktibilitas dan dicoverability dalam pengelolaan class library.

   a. *Capitalization Conventions*

      Terdapat tiga *style* dalam penamaan yang biasa digunakan, yaitu:

      1) *Pascal Casing*

         Pada *Pascal Casing* huruf pertama pada identifier dan huruf pertama pada kata berikutnya harus kapital.

Contoh : PropertyDescriptor

      2) *Camel Casing*

         Huruf pertama dari idetifier adalah huruf kecil dan huruf pertama pada kata berikutnya adalah huruf besar.

Contoh : propertyDescriptor

3) Uppercase

Semua huruf pada identifier adalah huruf kapital

Contoh : IO

Tabel II.1 Jenis-jenis Identifier

Identifier	**Casing**	**Example**
Class	Pascal	AppDomain
Enum type	Pascal	ErrorLevel
Enum values	Pascal	FatalError
Event	Pascal	ValueChange
Exception class	Pascal	WebException Note: Selalu gunakan Exception di akhir
Read-only Static field	Pascal	RedValue
Inteface	Pascal	IDisposable Note: Selalu gunakan awalan I
Method	Pascal	ToString
Namespace	Pascal	System.Drawing
Parameter	Camel	typeName
Property	Pascal	BackColor
Protected instance field	Camel	redValue
Public instance field	Pascal	RedValue

b. Untuk menghindari kebingungan dan menjamin operasi lintas bahasa, maka penggunaan case sensitif. Berikut adalah panduan penggunaan *case sensitivity* pada .NET Framework.

1) Jangan menggunakan nama yang membutuhkan *case-sensitive*. Komponen harus dapat digunakan dari kedua bahasa *case-sensitive*

dan *case-insensitive*. Bahasa *case-insensitive* tidak dapat membedakan antara dua nama dalam konteks yang sama.

2) Jangan membuat dua namespace dengan nama yang hanya dibedakan oleh *case* saja. Misalnya bahasa *case-insensitive* tidak dapat membedakan antara dua deklarasi namespace deklarasi berikut:

```
namespace ee.cummings;
namespace Ee.Cummings;
```

3) Jangan membuat fungsi dengan nama parameter yang hanya dibedakan oleh *case* saja. Contoh berikut adalah contoh yang salah dalam penggunaan nama parameter.

```
void MyFunction(string a, string A)
```

4) Jangan membuat namespace dengan nama tipe yang hanya dibedakan oleh *case* saja. Pada contoh berikut, point p dan POINT p adalah jenis penamaan yang tidak baik untuk digunakan.

```
System.Windows.Forms.Point p
System.Windows.Forms.POINT p
```

5) Jangan membuat *type* dengan nama properti yang hanya dibedakan oleh *case* saja. Pada contoh berikut, int Color dan int COLOR adalah penamaan properti yang tidak baik digunakan.

```
int Color {get, set}
int COLOR {get, set}
```

6) Jangan membuat type dengan nama method yang hanya dibedakan oleh case saja. Pada contoh berikut, *calculate* dan *Calculate* adalah penamaan method yang tidak baik untuk digunakan karena hanya dibedakan dengan *case*.

```
void calculate()
void Calculate()
```

c. General Naming Conventions

Terdapat hal-hal yang berhubungan dengan penamaan yang harus diperhatikan seperti pemilihan nama identifier harus mudah dibaca, penamaan yang ringkas dan mewakili, jangan menggunakan garis bawah, tanda hubung, atau karakter nonalphanumeric lainnya, dan jangan menggunakan notasi Hungaria.

Untuk menghindari kebingungan dan menjamin operasi lintas bahasa, ikuti aturan penggunaan singkatan dibawah ini:

1) Jangan gunakan singkatan sebagai bagian dari nama identifier. Misalnya, menggunakan GetWindow bukan GetWin.
2) Jangan menggunakan singkatan yang tidak umum dalam komputasi
3) Gunakan singkatan untuk menggantikan nama frase yang panjang. Misalnya menggunakan UI untuk User Interface.
4) Ketika menggunakan singkatan, gunakan pascal case dan camel case untuk singkatan yang karakternya lebih dari dua. Misalnya System.IO
5) Jangan menggunakan singkatan dalam identifiers atau nama parameter. Jika Anda harus menggunakan singkatan, gunakan camel case untuk singkatan yang terdiri dari lebih dari dua karakter, bahkan jika ini bertentangan dengan singkatan standar dari kata.

d. Class Naming Guidelines

Berikut ini adalah uraian pedoman untuk penamaan kelas:

1) Gunakan kata benda atau kata benda frase untuk nama kelas
2) Gunakan pascal case
3) Gunakan singkatan
4) Jangan menggunakan tipe awalan, seperti C untuk kelas dalam penamaan kelas, misalnya gunakan nama FileStream daripada CFileStream.

5) Jangan gunakan karakter garis bawah (_).
6) Terkadang diperlukan penamaan kelas yang diawali dengan karakter I meskipun kelas tersebut bukan Interface. Hal ini masih dianggap sesuai selama I adalah huruf pertama dari seluruh kata yang merupakan bagian nama kelas. Misalnya nama kelas IdentityStore.
7) Apabila diperlukan gunakan kata majemuk untuk nama kelas turunan. Bagian kedua dari nama kelas turunan harus menjadi nama kelas dasar. Misalnya, ApplicationException adalah nama kelas turunan yang tepat dari kelas yang bernama Exception, karena ApplicationException adalah jenis Exception.

```
public class FileStream
public class Button
public class String
```

e. Attribute Naming Guidelines

Dalam penamaan attribute selalu harus ditambahkan akhiran attribute untuk kelas attribute khusus. Berikut ini adalah contoh dari penamaan kelas attribute yang benar.

```
public class ObsoleteAttribute{}
```

f. Parameters Naming Guidelines

Berikut ini adalah uraian pedoman untuk penamaan parameter:

1) Gunakan camel case untuk penamaan parameter
2) Gunakan nama parameter deskriptif
3) Gunakan nama yang menggambarkan nama parameter dari pada nama-nama yang menggambarkan jenis parameter.
4) Jangan menggunakan nama parameter yang sudah ada
5) Jangan menggunakan notasi hungaria dalam penamaan parameter. Berikut ini adalah contoh dari penamaan parameter yang benar.

```
Type GetType(string typeName)
```

```
string Format(string format, object[] args)
```

g. Method Naming Guidelines

Aturan berikut menguraikan pedoman penamaan method:

1) Gunakan kata kerja atau frase kata kerja untuk nama method.
2) Gunakan pascal case. Berikut ini adalah contoh dari penamaan method yang benar.

```
RemoveAll()
GetCharArray()
Invoke()
```

h. Property Naming Guidelines

Aturan berikut menguraikan pedoman penamaan property:

1) Gunakan kata benda atau frase kata benda untuk penamaan property
2) Gunakan pascal case
3) Jangan menggunakan notasi hungaria
4) Pertimbangkan untuk membuat sebuah property dengan nama yang sama sebagai jenis yang mendasarinya. Berikut ini adalah contoh dari penamaan property yang benar.

```
public class SampleClass
{
   public Color BackColor
   {
      // Code for Get and Set accessors goes here.
   }
}
=======================================================
public enum Color
{
   // Insert code for Enum here.
}
public class Control
{
   public Color Color
```

```
    {
        get {// Insert code here.}
        set {// Insert code here.}
    }
}
```

i. Names of Assemblies and DLL's

Sebuah assembly mengamdung semua atau bagian-bagian dari library yang dapat digunakan kembali dan terdapat dalam sebuah single *dynamic-link library* (DLL). Sebuah assembly dapat dibagi menjadi multiple DLL tetapi hal ini jarang sekali digunakan. Berikut adalah panduan penamaan untuk DLL.

1) Jangan pilih nama DLL dengan nama yang menunjukan potongan besar fungsionalitas seperti System.Data. Assembly dan nama DLL tidak harus sesuai dengan nama sama *namespace* akan tetapi akan lebih baik mengikuti nama *namespace* saat penamaan assembly.
2) Pertimbangkan penamaan DLL sesuai dengan pola berikut.

```
<Company>.<Component>.dll
```

<Component> mengandung satu atau lebih klausa dot (titik) sebagai pemisah. Contoh penggunaannya adalah sebagai berikut.

```
Contoso.WebControls.dll
```

2. Type Design Guidelines

a. Types and Namespaces

Berikut panduan yang dapat membantu dalam mengatur jenis *types* dan *namespace*:

1) Hindari *namespace* yang terlalu banyak.
2) Jangan gunakan *namespace* untuk mengatur *types* menjadi hirarki daerah fitur yang terkait.
3) Hindari hirarki *namespace* yang sangat dalam.

4) Hindari desain *type* untuk skenario yang rumit dalam *namespace* yang sama sebagai tipe yang digunakan untuk tugas pemrograman yang umum.

b. Abstract Class Design

Karena *abstract* kelas sebaiknya tidak pernah dipakai, sangat penting untuk menentukan konstruktor yang benar untuk kelas *abstrak* tersebut. Berikut panduan agar kelas abstrak bekerja dengan benar pada saat diimplementasikan:

1) Jangan menentukan hak akses *public* atau *protected* internal pada konstraktor dalam tipe abstrak.
2) Jangan menentukan hak akses *protected* atau konstruktor internal dalam kelas abstrak.
3) Jangan menggunakan setidaknya satu tipe konkret yang mewarisi dari setiap kelas abstrak yang digunakan.

c. Static Class Design

Kelas statik merupakan kelas yang tidak mengandung *instance members* lain dari yang diwarisi dari *object*, dan tidak memiliki konstuktor *callable*. Berikut panduan agar kelas *static* dirancang dengan benar:

1) Jangan gunakan kelas *static* yang sederhana. Kelas *static* digunakan hanya sebagai pendukung kelas-kelas untuk *object oriented framework*.
2) Jangan mendeklarasikan atau melakukan *override instance member* pada kelas *static*. Jika desain kelas menunjukan harus ada *instance member*, kelasa seharusnya tidak ditandai dengan static.
3) Jangan mendeklarasikan kelas statis sebagai *sealed* dan *abstract*, dan tambahkan sebuah *private instance* konstuktor jika

bahasa pemrograman yang digunakan tidak memiliki built-in untuk mendukung kelas statis.

4) Jangan memperlakukan static class sebagai sebuah *miscellaneous bucket*. *Environment class* adalah contoh yang baik dari penggunaan yang tepat dari kelas statis. Kelas ini menyediakan akses ke informasi tentang lingkungan pengguna saat ini.

d. Interface Design

Berikut panduan untuk memastikan bahwa *interface* diancang dengan benar:

1) Jangan mendefinisikan sebuah *interface* jika kita perlu beberapa fungsi umum yang akan didukung oleh seperangkat *types* yang mencakup beberapa nilai *types*.
2) Pertimbangkan pemakaian *interface* jika kita butuh untuk mendukung fungsionalitas pada *types* yang sudah mewarisi dari *types* lainnya.
3) Hindari penggunaan *interface* tanpa anggota.
4) Sediakan paling sedikit satu *types* yang merupakan implementasi dari sebuah *interface*.
5) Sediakan paling sedikit satu anggota yang menggunakan setiap *interface*, contohnya sebuah *method* yang mengambil *interface* sebagai parameter atau *property*.

3. Member Design Guidelines

a. Member Overloading

Panduan berikut ini membantu memastikan bahwa member overloading dirancang dengan baik:

1) Jangan mencoba untuk menggunakan nama parameter deskriptif untuk menunjukan nilai default yang digunakan oleh overload sederhana. Pedoman ini berlaku untuk parameter Boolean. Semakin kompleks kelebihan nama parameter harus menunjukkan nilai default yang disediakan oleh overload sederhana dengan

menggambarkan keadaan atau tindakan yang berlawanan. Misalnya, kelas String menyediakan overloads berikut:

```
public static int Compare(
  string strA,
  string strB
);

public static int Compare(
  string strA,
  string strB,
  bool ignoreCase
);
```

2) Hindari melakukan penamaan parameter yang sembarangan pada overload. Jika satu parameter mempresentasikan inputan yang sams dengan overload lainnya, parameter harus memiliki nama yang sama. Untuk contoh jangan lakukan hal berikut ini:

```
public void Write(string message, FileStream
    stream){}
public void Write(string line, FileStream file, bool
    closeStream){}
```

Definisi yang benar untuk overloads ini adalah sebagai berikut:

```
public void Write(string message, FileStream
    stream){}
public void Write(string message, FileStream
    stream, bool closeStream){}
```

3) Parameter dengan nama yang sama akan muncul diposisi yang sama di semua overload. Untuk contoh jangan lakukan hal berikut ini:

```
public void Write(string message, FileStream
    stream){}
public void Write(FileStream stream,  string
    message, bool closeStream){}
```

Definisi yang benar untuk overloads ini adalah sebagai berikut:

```
public void Write(string message, FileStream
    stream){}
public void Write(string message, FileStream
    stream, bool closeStream){}
```

4) Jangan menggunakan ref atau out modifiers untuk member overload. Untuk contoh jangan lakukan hal-hal berikut ini:

```
public void Write(string message, int count)
...
public void Write(string message, out int count)
```

b. Property Design

Panduan berikut membantu memastikan properti Anda dirancang dengan baik:

1) Buat read-only properti jika pemanggil tidak harus mengubah nilai properti.
2) Jangan menyediakan set-only properti.
3) Berikan nilai default yang masuk akal untuk semua properti, memastikan bahwa default tidak mengakibatkan lubang keamanan atau desain yang sangat tidak efisien.
4) Memungkinkan properti untuk diatur dalam urutan apapun bahkan jika hal ini mengakibatkan keadaan sementara objek yang tidak valid.
5) Mempertahankan nilai sebelumnya jika setter properti melempar pengecualian.

6) Hindari membuang pengecualian dari Getter properti.

c. Parameter Design

Panduan berikut membantu memastikan properti dirancang dengan baik:

1) Jangan gunakan type parameter yang menyediakan fungsi yang diperlukan oleh anggota.
2) Jangan gunakan parameter reserved.
3) Jangan gunakan metode publik yang mengambil pointer, array dari pointer dan array multidimensi sebagai parameter.
4) Jadilah konsisten dalam penamaan parameter ketika meng-override anggota atau menerapkan antarmuka anggota.

4. Designing For Extensibility

a. Unsealed Clases

Pertimbangkan kelas unsealed tanpa anggota virtual atau protected sebagai cara yang bagus untuk memberikan kemurahan namun dihargai dalam ekstensibilitas framework.

b. Protected Members

Pertimbangkan untuk menggunakan anggota protected untuk kostumisasi, anggota protected merupakan cara yang baik untuk memberikan kostumisasi tanpa interface yang rumit.

c. Events and CallBacks

Berikut panduan penggunaan event dan callback:

1) Pertimbangkan untuk menggunakan callback untuk menyediakan custom code untuk dieksekusi framework.
2) Pertimbangkan penggunaan event untuk memungkinkan pengguna untuk menyesuaikan perilaku class library tanpa pengguna harus mengerti object oriented.
3) Hindari menggunakan callback di API.

d. Virtual Member

Anggota virtual berfungsi lebih baik dibandingkan dengan callback dan event, tetapi tidak lebih baik daripada metode non-virtual. Berikut panduan untuk perancangan virtual member:

1) Jangan membuat virtual member kecuali jika terdapat alasan yang baik untuk penggunaan dan dapat memelihara virtual member tersebut.
2) Berikan hak akses protected daripada public untuk virtual member.

e. Abstract Types and Interface

Abstract types dan interface menentukan abstraksi program, abstraki menentukan kontrak pewarisan yang harus dipatuhi. Jenis abstrak memiliki pilihan untuk memberikan rincian pelaksanaan sedangkan interface tidak dapat memberikan rincian pelaksanaan. Pertimbangkan penggunaan kelas abstrak atau interface saat merancang abstraksi.

5. Design Guidelines for Exceptions

a. Exception Throwing

Panduan berikut membantu memastikan Exception dirancang dengan baik:

1) Jangan mengembalikan errors codes. Exception adalah sarana utama pelaporan kesalahan dalam *framework*.

2) Lakukan pelaporan kegagalan ekseskusi dengan melemparkan exception. Jika anggota tidak dapat berhasil melakukan apa yang dirancang. Yang harus dipertimbangkan adalah kegagalan eksekusi dan exception harus dilemparkan

3) Pertimbangkan untuk menggunakan *exception builder methods*. Hal ini umum untuk melemparkan pengecualian yang sama dari tempat yang berbeda.

4) Hindari secara eksplisit melemparkan pengecualian dari akhir blok.

b. Exception Handling

Pedoman berikut membantu memastikan bahwa librarymenangani pengecualian tepat:

1) Tidak menangani kesalahan dengan menangkap pengecualian non-spesifik, seperti System.Exception, System.SystemException dan sebagainya, dalam *framework.*
2) Pertimbangkan catching specific exceptions ketika Anda memahami mengapa itu akan dilemparkan dalam konteks tertentu.
3) Tidak berlebihan menggunakan catch. Exception sering akan diizinkan untuk menyebarkan *up the call stack.*

6. Usage Guidelines

a. Array Usage Guidelines

1) *Array vs. Collection*

Desainer *class library* mungkin perlu membuat keputusan sulit mengenai kapan untuk menggunakan sebuah array dan kapan harus kembali menggunakan *collection*. Meskipun jenis ini memiliki model penggunaan yang sama, mereka memiliki karakteristik kinerja yang berbeda. Secara umum, Anda harus menggunakan *collection* ketika *add*, *remove*, atau *method* lain untuk memanipulasi *collection* yang didukung.

2) Properties that Return Arrays

Anda harus menggunakan collectios untuk menghindari inefisiensi kode yang disebabkan oleh sifat-sifat yang mengembalikan array. Dalam contoh kode berikut masing-masing panggilan ke properti myObj menciptakan salinan array. Sebagai akibatnya 2n + 1 salinan array akan dibuat dalam lingkaran berikut.

```
for (int i = 0; i < obj.myObj.Count; i++)
        DoSomething(obj.myObj[i]);
```

3) Returning Empty Arrays

Properti *string* dan *array* harus tidak mengembalikan *null reference*. Null bisa sulit untuk memahami dalam konteks ini.

Misalnya, pengguna mungkin menganggap bahwa kode berikut akan bekerja.

```
public void DoSomething()
{
    string s = SomeOtherFunc();
    if (s.Length > 0)
    {
        // Do something else.
    }
}
```

b. Attribute Usage Guidelines

Aturan berikut menguraikan pedoman penggunaan untuk kelas attribute:

1) Menambahkan akhiran atribut untuk kelas custom atribut, seperti yang ditunjukkan dalam contoh berikut:

```
public class ObsoleteAttribute{}
```

2) Menentukan AttributeUsage pada atribut untuk menentukan bahwa penggunaannya sudah tepat, seperti yang ditunjukkan dalam contoh berikut:

```
[AttributeUsage(AttributeTargets.All, Inherited = false,
    AllowMultiple = true)]
public class ObsoleteAttribute: Attribute {}
```

Segel atribut kelas bila memungkinkan, sehingga kelas tidak dapat diturunkan dari kelas tersebut.

II.6 Pengujian Perangkat Lunak

Pengujian perangkat lunak dilakukan untuk menguji seluruh fungsional yang telah dirancang. Dengan melakukan pengujian maka dapat diketahui kekurangan, error, bug yang terdapat pada program yang telah dibangun [14].

Pengujian perangkat lunak yang akan dilakukan pada penelitian ini menggunakan *unit testing*, pengujian integrasi, dan pengujian *acceptance*.

II.6.1 Unit Testing

Unit testing merupakan sebuah program yang digunakan untuk menguji sebuah method atau kelas tertentu. Unit test mengeksekusi method yang diuji menggunakan data sampel sebagai parameter dan memeriksa apakah method yang dieksekusi tersebut telah benar dan berjalan dengan baik. Pengujian unit testing dilakukan dengan menerapkan beberapa skenario pengujian yang berbeda [12].

II.6.2 Pengujian Integrasi

Pengujian Integrasi adalah pengujian yang dilakukan dengan mengintegrasikan suatu unit dengan unit yang lain yang saling berinteraksi sehingga menyusun suatu sistem. Pengujian integrasi juga digunakan untuk mengetahui kesalahan dan masalah yang terdapat pada perangkat lunak [15].

II.6.3 Pengujian Acceptance

Pengujian acceptance dilakukan untuk menentukan apakah perangkat lunak yang dibangun telah sesuai dengan tujuan. Pengujian ini memberikan kepercayaan kepada pengguna bahwa perangkat lunak yang dibuat telah sesuai dengan apa yang dibutuhkan oleh pengguna .

II.7 Windows Phone

Microsoft Windows Phone adalah platform mobile yang berisi fitur-fitur modern termasuk GPS, e-mail, SMS, Kamera, dan *music player* serta menyediakan framework pembangunan yang mudah sehingga memungkinkan untuk jutaan pengembang .NET untuk belajar dan mengembangkan pada Windows Phone dengan cepat. Windows Phone dalam pengembangannya menggunakan bahasa C# dan Visual Basic. Sebagai framework aplikasi dapat menggunakan Silverlight atau XNA. Silverlight menggunakan Extensible Application Markup Language yaitu

sebuah bahasa markup deklaratif untuk menciptakan User Interface yang menarik untuk Windows Phone. Dengan menggunakan tools seperti Dobe Ilustrator, Photoshop, dan Microsoft Expression Design, pengembang dapat membuat UI berbasis vektor yang dapat diekspor ke dalam XAML. XNA digunakan dalam menciptakan game. Framework ini dilengkapi dengan game engine yang memungkinkan untuk membuat game berabasis-loop dan 3D engine yang digunkan untuk mebangun game 3D [16].

II.8 Analisis Domain

Analisis domain dilakukan dengan menganalisis beberapa aplikasi sejenis dengan tujuan untuk mengetahui kesamaan fungsional yang terdapat pada setiap aplikasi. Analisis domain dilakukan dengan beberapa tahap yaitu analisis frozen spot, analisis hotspot dan pendefinisian kartu hotspot.

II.8.1 Frozen Spot

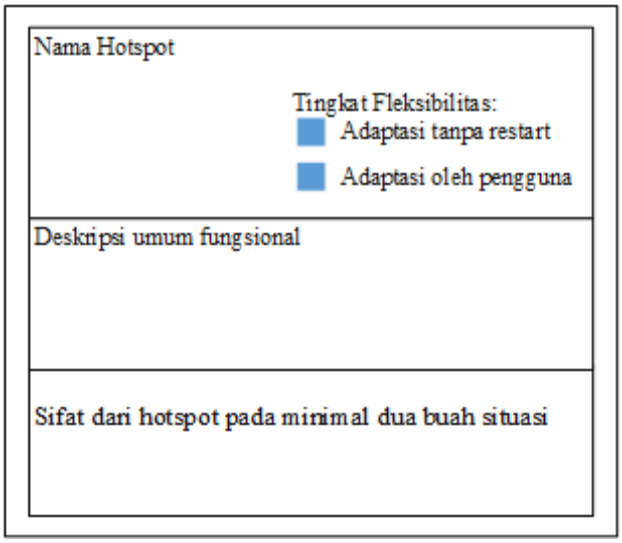
Frozen spot merupakan fungsi-fungsi yang tetap yang terdapat pada aplikasi dengan domain kasus tertentu [17]. Frozen Spot dapat disebut irisan dari fungsional dasar yang terdapat dalam aplikasi. Fungsi-fungsi pada setiap aplikasi pada domain aplikasi tertentu dianalisis. Hasil analisis dari frozen spot kemudian akan digunakan untuk menentukan hotspot yang terdapat pada aplikasi sejenis.

II.8.2 Hotspot

Hotspot merupakan titik fleksibilitas dari aplikasi yang artinya fungsional yang pasti terdapat pada setiap aplikasi pada domain kasus tertentu [17]. Analisis hotspot dapat diuraikan dari frozen spot. Hotspot yang diambil merupakan sebagai dasar dari aplikasi dan tidak dapat dihapus. Hotspot dirancang untuk lebih spesifik dan dapat disesuaikan dengan kebutuhan domain [18]. Terdapat duah metode untuk mendefinisikan hotspot, yaitu metode *blackbox* dan metode *whitebox* [19]. Metode *blackbox* yaitu metode untuk menentukan hotspot bedasarkan hasil analisis dari frozen spot. Pada metode *blackbox* fungsional yang dapat dijadikan menjadi hotspot harus terdapat pada setiap aplikasi pada domain aplikasi tertentu. Sedangkan menentukan hotspot dengan menggunakan metode *whitebox* dilakukan berdasarkan kebutuhan dari hotspot lainnya.

II.8.3 Kartu Hotspot

Kartu Hotspot digunakan untuk mendeskripsikan fungsionalitas dari setiap hotspot yang ada [20]. Kartu hotspot terdiri dari beberapa bagian yaitu nama hotspot, derajat fleksibilitas, deskripsi fungsional, dan perilaku hotspot terhadap dua situasi yang berbeda. Nama kartu hotspot harus menggambarkan fungsionalitas. Pada derajat fleksibilitas terdapat dua pilihan yaitu adaptasi tanpa restart dan adaptasi oleh end user. Adaptasi tanpa restart berarti adaptasi dari hotspot dapat langsung digunakan sedangkan adaptasi oleh end user berarti hotspot tersebut harus diadaptasi oleh end user sebelum dapat digunakan.



Gambar II.1 Contoh Gambaran Kartu Hotspot

II.9 Alat Pemodelan

Alat pemodelan merupakan alat yang digunakan untuk memodelkan konsep-konsep dan perancangan yang nantinya akan diimplementasikan. Alat pemodelan yang akan digunakan pada penelitian ini adalah UML.

II.9.1 UML

UML(Unified Modeling Language) merupakan teknik pemodelan yang memudahkan pengembang untuk memspesifikasikan, memfisualisasikan, dan mendokumentasikan model dengan suatu cara yang mendukung skalabilitas, keamanan, dan eksekusi. UML memudahkan untuk mengidentifikasi pola-pola

prilaku pada sistem. UML dapat digunakan dalam pembutan desain yang modular sehinggaa mempercepat pembangunan suatu sistem [21].

1. *Use Case Diagram*

   Use Case Diagram merupakan gambaran dari fungsi-fungsi utama yang terdapat dalam sistem [22]. Use Case juga menjelaskan keterkaitan antara sistem dengan kebutuhan perangkat lunak dan menggambarkan apa yang harus sistem lakukan.

2. *Class Diagram*

   Class diagram merupakan salah satu pemodelan pada UML yang menggambarkan struktur kelas pada sistem. Class diagram mendeskripsikan tipe-tipe objek yang digunkan dalam sistem dan keterhubungan antar kelas. Setiap objek pada kelas memiliki hak akses terdapat tiga jenis yaitu protected, public, dan private [22].

3. *Package Diagram*

   Package diagram atau diagram paket merupakan model yang digunakan untuk mengelompokan kelas. Diagram paket digunakan untuk melihat ketergantungan antar paket [22]. Package diagram digunakan untuk mengetahui ketergantungan antar bagian dari suatu perangkat lunak, mencari masalah pada perangkat lunak, dan menentukan urutan kompilasi.