**Template**

**Outline**

**Description**

**1. Introduction**  
1.1 Object design trade-offs  
1.2 Interface documentation guidelines  
1.3 Definitions, acronyms, and abbreviations  
1.4 References

*The first section of the ODD is an introduction to the document. It describes the general trade-offs made by developers (e.g., buy vs. build, memory space vs. response time), guidelines and conventions (e.g., naming conventions, boundary cases, exception handling mechanisms), and an overview of the document.  
Interface documentation guidelines and coding conventions are the single most important factor that can improve communication between developers during object design. These include a list of rules that developers should use when designing and naming interfaces. These are examples of such conventions:*

• Classes are named with singular nouns.

• Methods are named with verb phrases, fields, and parameters with noun phrases.

• Error status is returned via an exception, not a return value.

• Collections and containers have an elements() method returning an Enumeration.

*• Enumerations returned by elements() methods are robust to element removals.*

Such conventions help developers design interfaces consistently, even if many developers contribute to the interface specification. Moreover, making these conventions explicit before object design makes it easier for developers to follow them. In general, these conventions should not evolve during the project.

**2. Packages**

*The second section of the ODD, Packages, describes the decomposition of subsystems into packages and the file organization of the code. This includes an overview of each package, its dependencies with other packages, and its expected usage.*

**3. Class interfaces  
Glossary**

*The third section, Class interfaces, describes the classes and their public interfaces. This includes an overview of each class, its dependencies with other classes and packages, its public attributes, operations, and the exceptions they can raise.*

Progetto di Ingegneria del software

Anno accademico 2018/2019

*Carcheck*

**Tabella Componenti**

**Capriglione**

**Francesco**

**D’Auria**

**Aldo**

**De Falco**

**Daniele**

0512104666

**Iacovazzo**

**Giovanni**

0512104774

**Indice**

**1. Introduzione**

1.1 Object design trade-off

1.2 Interface documentation guidelines

1.3 Definizioni, acronimi e abbreviazioni

1.4 Riferimenti

**2. Packages**

**3. Class Interface Glossary**

***1.0 Introduzione***

Nel seguente documento vengono definiti tutti gli aspetti analizzati prima di procedere all’implementazione. Nelle fasi precedenti sono stati definiti i vari requisiti su cui modellare il sistema CarCheck. Il nostro scopo qui è andare ad interfacciarci con i Solution Objects, e tenere conto di aspetti di aspetti significativi riguardanti tempo di esecuzione di un’operazione, utilizzo di memoria e altre misure di costo.

***1.1 Object design trade-off***

***Interfaccia vs Usabilità***

L’interfaccia, dovendo essere semplice da utilizzare ed intuitiva, utilizzerà una serie di componenti standard. Saranno inoltre presenti delle etichette, suggerimenti ed altri elementi che faranno capire all’utente che l’azione che sta eseguendo ha prodotto un risultato oppure è incorretta. Eventualmente potranno essere utilizzate delle legende per una maggiore compresione di simboli e notazioni.

***Spazio di memoria vs Tempo di risposta***

Si cercherà di mantenere un equilibrio tra questi due parametri. Infatti, da una parte abbiamo una grande quantità di informazioni da memorizzare su un numero molto alto di veicoli, dall’altra bisogna assicurare che i tempi delle risposte alle richieste degli utenti rispettino ciò che è stato espresso nella definizione dei requisiti non funzionali.

***Sicurezza vs efficienza***

Per gli utenti registrati sarà data priorità alla sicurezza. L’approccio utilizzato sarà quello di utilizzare come fattore di autenticazione una coppia Email-Password per evitare accessi non autorizzati. Altro aspetto, sempre relativo alla sicurezza, riguarda la criptazione dei dati sensibili trasmessi al server (email, password, alcuni dati relativi a revisioni).

Per gli utenti non registrati tali problematiche non sono affatto presenti. La priorità è quindi data interamente all’efficienza.

***Prestazioni vs Costi:***

Essendo un progetto universitario e non avendo risorse finanziarie, tutte le tecnologie da noi scelte sono distribuite sotto licenza open-source o free-ware. Per questo motivo i costi non sono un aspetto da tenere in considerazione.

Per quanto riguarda invece le prestazioni, tali tecnologie forniscono comunque dei buoni risultati sotto diversi aspetti e per questo rappresentano una buona soluzione per la realizzazione del sistema.

***1.2 Interface documentation guidelines***

Durante l’implementazione del progetto è necessario che siano seguite le guide linea qui sotto esposte.

È anche importante commentare ciò che viene fatto nel codice, tramite normali commenti o strumenti come ad esempio Javadoc che permette di ottenere una vera e propria documentazione.

***Convenzioni sui nomi***

I nomi utilizzati per la rappresentazione dei concetti principali, delle funzionalità e delle componenti generiche del sistema devono rispettare le seguenti condizioni:

Quando si utilizzano dei nomi per rappresentare componenti, concetti, operazioni o funzionalità di CarCheck bisogna stabilire che essi:

• Non devono avere una lunghezza “eccessiva”

• Composti da lettere dell’alfabeto italiano\anglo-sassone.

• Composti eventualmente da caratteri numerici

Per i nomi riguardanti elementi della programmazione, sono presenti delle convenzioni più rigide.

Una prima distinzione viene fatta nella notazione tra variabili e costanti.

Una seconda distinzione viene fatta nella notazione tra classi, interfacce e package.

Per rappresentare le ***variabili*** bisogna:

• Utilizzare caratteri minuscoli, o almeno evitare che il primo carattere sia maiuscolo.

• Non utilizzare caratteri speciali ( ad esempio #,@,$,% )

Riassumendo si vuole evitare la Pascal Notation ed utilizzare invece la Camel Notation.

Questo è un esempio corretto:

Mentre questo è un esempio di errore:

Per le ***costanti*** le convenzioni da seguire sono:

• Utilizzare soltanto caratteri maiuscoli

• Per separare due parole distinte utilizzare il carattere underscore ( “\_” )

• Non utilizzare il carattere underscore all’inizio di una parola o parola composta

• Non utilizzare caratteri speciali ( ad esempio #,@,$,% )

Altre convenzioni riguardano le classi ( e relativi metodi ), le interfacce e i packages.

***Classe***

• Dovrebbe iniziare con una lettera maiuscola

• Dovrebbe essere un ***sostantivo*** ( Ad esempio Color, Button, System ecc. )

• Evitare di utilizzare acronimi

***Interfaccia***

• Dovrebbe iniziare con una lettera maiuscola

• Dovrebbe essere un ***aggettivo*** (Ad esempio Runnabl , Remote, ActionListener ecc. )

• Evitare di utilizzare acronimi

***Package***

• Dovrebbe essere scritto in minuscolo

• Se esso contiene più parole, ciascuna di esse deve essere separata da un punto “.”

***Metodo***

• Dovrebbe iniziare con una lettera minuscola

• Dovrebbe essere un ***verbo*** (print(), doSomething() );

• Se il nome contiene più parole, la prima parola dovrebbe iniziare con una lettera minuscola mentre le altre con una lettera maiuscola

• I nomi get() , set() sono riservati ai metodi getters and setters.

**2. Packages**

Classe:

Descrizione:

AdhesionRequest.java

Descrive un utente registrato al sistema

AdminBean.java

Descrive un admin all’interno del sistema

CityBean.java

Descrive tutte le informazioni di una città

InsuranceBean.java

Descrive tutte le informazioni relative all’assicurazione su un veicolo

InsuranceCompanyBean.java

Descrive tutte le informazioni relative ad una compagnia assicurativa

OwnerBean.java

Rappresenta l’unione tra il veicolo e il proprietario del veicolo

PeopleBean.java

Descrive la persona esterna proprietaria di un veicolo

VehicleInspectionBean.java

Rappresenta tutte le informazioni sulla ispezione di un veicolo

ProvinceBean.java

Rappresenta tutte le informazioni di una provincia

RegionBean.java

Rappresenta tutte le informazioni di una regione

PossessionFeeBean.java

Rappresenta la tassa di possesso su una vettura

VehicleBean.java

Rappresenta la struttura di un veicolo

WorkShopBean.java

Rappresenta la struttura di una officina

**3. Glossario**

**Query**: In informatica il termine **query** viene utilizzato per indicare l'interrogazione da parte di un utente di un database, strutturato tipicamente secondo il modello relazionale, per compiere determinate operazioni sui dati

**Cookie**: Piccoli file di testo inviati da un web server al browser dell’utente e che vengono memorizzati sul dispositivo di quest’ultimo con lo scopo di identificarlo.

**Browser**:Programma che consente di navigare ed interagire con le pagine web, i testi, le immagini ed altri elementi multimediali che formano internet o una rete locale.