



**Università degli Studi di Napoli Federico II**

---

Scuola Politecnica e delle Scienze di Base  
Corso di Laurea Magistrale in Ingegneria Aerospaziale

TESI DI LAUREA MAGISTRALE  
IN  
INGEGNERIA AEROSPAZIALE

## **Development of a Java-Based Framework for Aircraft Preliminary Design**

Wing Aerodynamic Analysis Module,  
Aircraft Longitudinal Static Stability Module

Relatori:

**Prof. Ing. Fabrizio Nicolosi**  
**Prof. Ing. Agostino De Marco**

Candidato:

**Manuela Ruocco**  
**Matricola M53/408**



*Dedica*

# ABSTRACT

The thesis deals with the development of adopt), a Java-based application conceived as a fast, reliable and user friendly computational aid for aircraft designers in the conceptual/preliminary design phases of a transport aircraft. The ultimate goal of such a software framework is to perform a multi-disciplinary analysis of an aircraft and then search for an optimized configuration. The search domain boundaries are usually defined by the user through a set of specified parameters.

Currently, the software is able to estimate the aircraft weight breakdown, the center of gravity location, the main aerodynamic parameters, and some stability derivatives. All these types of estimates can be usually performed using several interchangeable analysis methods. The wing aerodynamic load has been in particular estimated with a numerical method taken from nasablackwell. An extensive work has been carried out to validate all the results returned by the application.

ADOpT can be used from the command line or with a dedicated graphical user interface (GUI). The GUI allows the user to have an immediate feedback about the aircraft features when changing the input parameters, to manage multiple aircraft simultaneously and compare them side by side, and to view a 3D CAD model of the aircraft. The CAD model has been generated using the well known opencascade libraries, and can be eventually exported to file for external usage (e.g., in CAD/CAE suites).

Regarding the Input/Output capability of adopt, the application accepts configuration files in XML (eXtensible Markup Language) format and exports the results on file in two possible formats: XML and Microsoft Excel (XLS). The XML input files are also used in interactive work sessions to import a predefined aircraft and populate the GUI controls accordingly. The application has been developed making extensive use of the latest Java features introduced in 2014, i. e. the Java 8 release by Oracle. This release includes the JavaFX platform, which has been used to set up the 3D view.

# SOMMARIO

La tesi descrive lo sviluppo di *diadopt*, un'applicazione scritta in linguaggio Java concepita per essere uno strumento veloce, affidabile e di semplice utilizzo per le fasi di sviluppo concettuale e preliminare di un velivolo da trasporto. Lo scopo finale del programma è effettuare un'analisi multidisciplinare di una configurazione definita dall'utente e successivamente alterarla in modo da ottenerne una ottimizzata. Il dominio di ricerca di tale configurazione è definito dall'utente tramite un apposito insieme di parametri.

Allo stato attuale il programma effettua la stima dei pesi dei principali componenti di un velivolo, valuta la posizione del baricentro, i principali parametri aerodinamici e alcune derivate di stabilità. La stima di ciascun parametro può essere generalmente effettuata tramite diversi metodi tra loro intercambiabili. Il carico aerodinamico sull'ala, in particolare, è stato stimato tramite un metodo numerico tratto da NASA Blackwell. Molta attenzione è stata in generale dedicata alla validazione dei risultati forniti dall'applicazione.

*ADOpT* può essere usato sia da riga di comando sia tramite un'apposita interfaccia grafica (GUI). Quest'ultima fornisce un riscontro immediato riguardo il cambiamento delle prestazioni del velivolo nel momento in cui l'utente modifica uno o più parametri di input; inoltre permette di gestire più velivoli (o più configurazioni dello stesso velivolo) simultaneamente, di confrontarne le caratteristiche e di visualizzarne il modello CAD. Questo è generato tramite la libreria *opencascade* e può essere salvato su file in modo da poterlo usare in altri applicativi.

Per quanto riguarda i files di input e di output, l'applicazione accetta files in formato XML (*eXtensible Markup Language*) contenenti la configurazione del velivolo e può esportare i risultati sia in formato XML sia in formato XLS. I file XML di uscita possono essere successivamente re-importati e quindi modificati tramite l'interfaccia grafica. L'applicazione è stata sviluppata facendo largo uso delle ultime caratteristiche del linguaggio Java introdotte da Oracle nel 2014 con la versione 8; questa include, tra l'altro, la piattaforma *JavaFX* che è stata usata per costruire il visualizzatore del modello CAD.

---

# Contents

---

<b>I</b>	<b>Aircraft Design Overview</b>	<b>1</b>
1	Aircraft Design	2
2	ADOpT applicaton overview	3
<b>II</b>	<b>Development of Application</b>	<b>4</b>
3	Introdution to Java	5
4	Work Object	6
4.1	Introduction . . . . .	6
4.2	Input data from .XML file . . . . .	6
4.3	Default Aircraft . . . . .	7
4.3.1	How is made a default Aircraft . . . . .	8
4.3.2	How is made a default Wing . . . . .	11
4.4	Database in JPAD . . . . .	12
4.4.1	Setup database . . . . .	12
4.4.2	Assign database using an Aircraft object . . . . .	13
4.4.3	Assign database using a Wing object . . . . .	13
4.4.4	Developer's guide . . . . .	14
<b>III</b>	<b>Functionality Overview</b>	<b>16</b>
5	Lift characteristics of a wing	17
6	Drag characteristics of a wing	18
7	Longitudinal stability of an aircraft	19
8	Minor Works	20
<b>Appendices</b>		
<b>A</b>	<b>Java Reference</b>	<b>22</b>
A.1	Inheritance, Overriding and Polymorphism . . . . .	22
A.2	Packages . . . . .	22
A.3	Documentation . . . . .	23

---

<b>Glossary</b>	<b>25</b>
<b>Acronyms</b>	<b>27</b>
<b>List of symbols</b>	<b>28</b>
<b>Bibliography</b>	<b>30</b>

# Part I

## Aircraft Design Overview



# Chapter **1**

---

## Aircraft Design

---

# Chapter **2**

---

## ADOpT applicaton overview

---

## Part II

# Development of Application

# Chapter **3**

---

## Introduction to Java

---

---

## Work Object

---

*Citazione  
citazione  
– Autore*

### 4.1 Introduction

In JPAD it is possible to read an .XML file as input or generate an object whose data are written in the code. Both in the first and in the second case all needed variables are initialized with data relating the choosen aircraft. The difference between these two methos is that using an .XML file, user can to define its own aircraft having clear view about the needed data useful for the analysis.

Contrariwise in order to perform test of program functionality, to use a default aircraft is the most simple way to generate a work object.

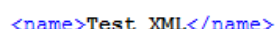
### 4.2 Input data from .XML file

XML is a file extension for an *Extensible Markup Language (XML)* file format used to create common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere using standard ASCII text. It is defined “Markup Language” due to the use of tags that describes the content. XML is considered extensible because the markup symbols are unlimited and self-defining. So it is possible to use personal tag for each data. In this way to read an .XML file results relatively simple .

The key concepts of an .XML File Format are the followings:

- markup symbol (tag)
- attribute
- tree structure

As mentioned, each part of the test is contained between an opening **markup symbol** and an end markup symbol that expressed the meaning of the text.



```
<name>Test XML</name>
```

Figure 4.1: Use of markup symbols in XML language.

In addition to tag name, the markup symbols may contain also some **attributes** that introduce more informations such as the unit of measure.

```
<tag attribute1='value' attribute2='value'> text </tag>
```

Figure 4.2: Use of attributes in XML language.

An .XML file has a tree structure where there are external knots that branch into internal knots.

In order to read an XML file it is necessary, first of all, to give the file path. The class `JPADXMLReader` opens the file and the methods of the class `MyXMLReaderUtils` reads the useful data from the XML having the tag path as input. It is possible to read data as Amount, namely with units of measurement or as double. The unit of measurement is written in the attributes of data in XML file.

Likewise it is possible to write output data on XML file using `JPADDataWriter` class. First of all it is necessary to define and build the xml tree structure. After each variable is associated to a name that is the markup symbol of the XML file.

## 4.3 Default Aircraft

Actually it is possible to define two different aircraft in order to test the functionality of the application: **ATR-72** and **B747-100B**.

The **ATR 72** is a twin-engine turboprop made by the French-Italian aircraft manufacturer ATR entered service in 1989. It was developed as a variant of the ATR 42 with a 4.5 m stretched fuselage. The ATR 72 was developed from the ATR 42 in order to increase the seating capacity (48 to 66 in standard configuration) by stretching the fuselage, increasing the wingspan, adding more powerful engines, and increasing fuel capacity by approximately 10 percent.[4] It has been typically employed as a regional airliner, although other roles have been performed by the type such as corporate transport, cargo aircraft and maritime patrol aircraft. [5]

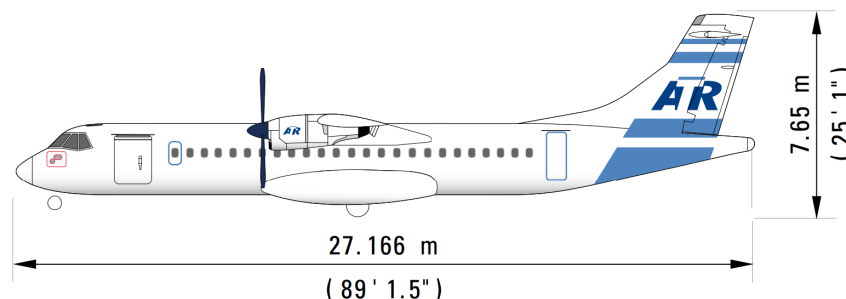


Figure 4.3: ATR 72. Side view.

The **Boeing 747-100B** is a four-engined long-range widebody commercial jet airliner and cargo aircraft produced by the American manufacturer Boeing Commercial Airplanes. It has a capacity of maximum 480 passengers in a partial double deck configuration. The Boeing 747 It is also known as Jumbo Jet. The basic B747-100 entered service with Pan American On January 15, 1970.

One of the reason to create the 747 was reductions in airfares with a consequent increase of passenger traffic[17]. The original version of the 747 had two and a half times greater capacity than the Boeing 707, one of the common large commercial aircraft of the 1960s and it was the largest passenger carrier from 1970 until the introduction of Airbus A380.[16] The Boeing 747 had two aisle and four wing-mounted engines. The upper deck is its distinctive "hump" along the forward part of the aircraft. It provides space for a lounge or extra seating. The raised cockpit allows front loading of cargo on freight variants.

The 747-100B model was developed from the 747-100SR. This configuration had a typical 452 passengers and unlike the original 747-100, the 747-100B was offered with Pratt & Whitney JT9D-7A, General Electric CF6-50, or Rolls-Royce RB211-524 turbofan engines.

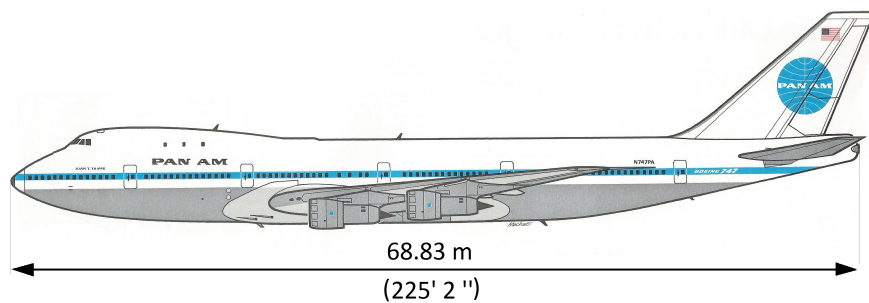


Figure 4.4: Boeing 747-100B. Side view.

### 4.3.1 How is made a default Aircraft

In order to define a Default Aircraft in a test class, and use it to check the functionalities of the application, it is necessary to follow some step. First of all it is necessary to initialize the working directory tree using the method `initWorkingDirectoryTree` of `MyConfiguration` class located in `JPADConfigs` package that initializes the working directory tree and fill the map of folders. This step is required in order to create the following default folders that are necessary for the right behavior of the code:

- Database directory
- Input directory
- Output directory

Using `MyConfiguration` class it's possible to point at a specific folder, like the input or output directory, with the static method `getDir`. This is a crucial step that must be execute at the beginning of every test. To set the working directory with the useful folders, it's necessary to call the function `initWorkingDirectoryTree()` at the beginning of each test. The function creates all necessary folders. Moreover the function has been overloaded and it can be even called with a variable number of arguments (`initWorkingDirectoryTree( String...str)`). These strings are the directory strings in `MyConfiguration` class. After it is possible to create an `Aircraft` object choosing between "ATR-72" or "B747-100B" using the method `createDefaultAircraft` from `Aircraft` class. This method defines a new `Aircraft` object and invokes another `Aircraft`'s methods that creates the component using default data. In the method `createDefaultAircraft` there is a calling to the

builder of `Aircraft` class that initializes the objects of the classes that perform calculations. At this step all the components of the aircraft are created. It is possible also to define new airfoil for the aircraft or change some data from the existing.

Afterwards it is necessary to set the operating conditions such as the number of Mach of analysis or altitude. Each default aircraft has a set of default condition but the user could to change them.

In order to manage all the aircraft related analysis it is necessary to define an object of the class `ACAnalysisManager`. Similarly to the aircraft, exist an analysis manager also for the wing that is an object of the `LSAerodynamicAnalysis` class.

The next step is to define and assign the needed databases. This will be explained in detail in the next section. Finally it is possible to do analysis.

### Listing 4.1 Generation of default aircraft

```
public static void main(String[] args) {

    // -----
    // Define directory
    // -----
    MyConfiguration.initWorkingDirectoryTree();

    // -----
    // Generate default Aircraft
    // -----
    Aircraft aircraft = Aircraft.createDefaultAircraft("B747-100B");
    LiftingSurface theWing = aircraft.get_wing();

    // Default operating conditions
    OperatingConditions theConditions = new OperatingConditions();

    // -----
    // Define an ACAnalysisManager Object
    // -----
    ACAnalysisManager theAnalysis = new ACAnalysisManager(theConditions);
    theAnalysis.updateGeometry(aircraft);

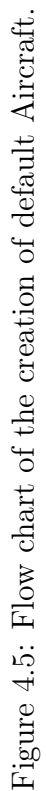
    // -----
    // Define an LSAerodynamicsManager Object
    // -----
    LSAerodynamicsManager theLSAnalysis = new LSAerodynamicsManager (
        theConditions,
        theWing,
        aircraft
    );

    // -----
    // Setup database(s)
    // -----

    theLSAnalysis.setDatabaseReaders(
        new Pair(DatabaseReaderEnum.AERODYNAMIC,
            "Aerodynamic_Database_Ultimate.h5"),
        new Pair(DatabaseReaderEnum.HIGHLIFT,
            "HighLiftDatabase.h5")
    );

    // -----
    // Do analysis
    // -----
    theAnalysis.doAnalysis(aircraft,
        AnalysisTypeEnum.AERODYNAMIC);
}
```





### 4.3.2 How is made a default Wing

Similarly to the default aircraft it is possible to define a default wing. This is very useful if the user wants to make an analysis only on a wing. In this case it is necessary to define the origin of the **Local Reference Frame (LRF)** in **Body Reference Frame (BRF)** and the coordinates of the **Gravity Center (GC)**.

Contrary to the case of the aircraft, for an isolated wing there isn't necessary to define a fuselage in order to create a **Lifting Surface** object, but there is an overload of the builder that doesn't need a fuselage as input. In this case the exposed surface is calculated as the surface of the wing.

**Listing 4.2** Generation of an isolated Wing

```
public static void main(String[] args) {

    // Assign all default folders
    MyConfiguration.initWorkingDirectoryTree();

    // -----
    // Coordinates of LRF
    // -----

    double xAw = 11.0; //meter
    double yAw = 0.0;
    double zAw = 1.6;
    double iw = 0.0;

    // -----
    // Generate default Wing
    // -----

    LiftingSurface theWing = new LiftingSurface(
        "Wing", // name
        "Data_from_AC_ATR_72_REV05.pdf",
        xAw, yAw, zAw, iw,
        ComponentEnum.WING
    );

    theWing.calculateGeometry();
    theWing.getGeometry().calculateAll();

    // -----
    // Center of Gravity
    // -----

    double xCgLocal= 1.5; // meter
    double yCgLocal= 0;
    double zCgLocal= 0;

    CenterOfGravity cg = new CenterOfGravity(
        Amount.valueOf(xCgLocal, SI.METER), // coordinates in LRF
        Amount.valueOf(yCgLocal, SI.METER),
        Amount.valueOf(zCgLocal, SI.METER),
        Amount.valueOf(xAw, SI.METER), // origin of LRF in BRF
        Amount.valueOf(yAw, SI.METER),
        Amount.valueOf(zAw, SI.METER),
        Amount.valueOf(0.0, SI.METER), // origin of BRF
        Amount.valueOf(0.0, SI.METER),
        Amount.valueOf(0.0, SI.METER)
    );

    cg.calculateCGinBRF();
    theWing.set_cg(cg);
    theWing.set_aspectRatio(6.0);

    // Default operating conditions
    OperatingConditions theOperatingConditions = new OperatingConditions();
    theOperatingConditions.set_alphaCurrent(Amount.valueOf(2.0, NonSI.DEGREE_ANGLE))

    // -----
    // Define an LSAerodynamicsManager Object
```

```

// -----
LSAerodynamicsManager theLSAnalysis = new LSAerodynamicsManager (
    theOperatingConditions,
    theWing
);

// -----
// Setup database(s)
// -----

theLSAnalysis.setDatabaseReaders(
    new Pair(DatabaseReaderEnum.AERODYNAMIC,
              "Aerodynamic_Database_Ultimate.h5"),
    new Pair(DatabaseReaderEnum.HIGHLIFT, "HighLiftDatabase.h5")
);

// -----
// Assign Airfoil(s) ...
// -----

// Define airfoilRoot...

// -----
// Set Airofoil(s)
// -----
List<MyAirfoil> myAirfoilList = new ArrayList<MyAirfoil>();
myAirfoilList.add(0, airfoilRoot);
myAirfoilList.add(1, airfoilKink);
myAirfoilList.add(2, airfoilTip);
theWing.set_theAirfoilsList(myAirfoilList);
theWing.updateAirfoilsGeometry();
theLSAnalysis.initializeDependentData();
}

```

## 4.4 Database in JPAD

In JPAD it is possible to consult external databases in .h5 format. **HDF 5** (Hierarchical Data Format Release 5) is a data file format designed by the *National Center for Supercomputing Applications* (NCSA) to assist users in the storage and manipulation of scientific data across different operating systems and machines.

To obtain the useful data in JPAD interpolating functions are used. These functions can be of one, two or three dimensions and read data from graphics that have been digitize previously.

Starting from these digitalizations, databases in .h5 format are built. Reading data from databases is entrusted to methods of classes in the database package.

In order to read these databases, and obtain the useful data, it is necessary to define an object of the database reading class and associate it with the object of analysis.

This is a crucial step to read correctly the external data. In fact JPAD allows to work with an aircraft object or only with an isolated lifting surface object. Aircraft is usually composed of a fuselage, lifting surfaces, nacelle and power plant. Furthermore, Aircraft and Wing are associated with classes of calculation like LSAerodynamicManager or ACAnalysisManager. So it is necessary that these databases are also visible from these classes.

So because both in aircraft and in wing there is a lifting surface object, databases relative to wing are associated to LSAerodynamicManager.

### 4.4.1 Setup database

Here the database path it's created and associated to object that interpolates the required data from the .h5 file using a MyInterpolatingFunction object. After

this it's possible to access the double value of the interpolating function using the `standaloneutils` method called `value`.

Now the procedure to assign the database is different if is used an `Aircraft` object or a `Wing` object.

#### 4.4.2 Assign database using an Aircraft object

In order to assign correctly the database and associate it to all analysis management is necessary to practise the following order.

1. Define an `Aircraft` Object.  
This command associates to `Aircraft` an object that defines the aerodynamic. From the wing it is possible to obtain the `Wing`, that is a `LiftingSurface` object.
2. Define an `ACAnalysisManager` object.  
All the aircraft computations are managed by this class.
3. Define an `LSAerodynamicManager` object.  
All the lifting surfaces computations are managed by this class.
4. Associate database to `LSAerodynamicManager`.
5. Eventually do analysis.

#### 4.4.3 Assign database using a Wing object

Using a `Wing` object it isn't necessary to define a manager for `Aircraft` aerodynamic analysis. So the step to follow are the same of aircraft starting from the third.

1. Define an `Wing` Object.
2. Define an `LSAerodynamicManager` object.
3. Associate database to `LSAerodynamicManager`.

The definition of a isolated `Wing` is explained in the relative section.

#### Listing 4.3 Assign database using an Aircraft object

```
// -----  
// Setup database(s)  
// -----  
  
theLSAnalysis.setDatabaseReaders(  
    new Pair(DatabaseReaderEnum.AERODYNAMIC,  
              "Aerodynamic_Database_Ultimate.h5"),  
    new Pair(DatabaseReaderEnum.HIGHLIFT,  
              "HighLiftDatabase.h5")  
);
```

The databases are assigned to `LSAerodynamic` using a method of this class. This method accept as input a variable number of `Pair` objects. Using `Pair` objects it is possible to assign, for each database, both name and type.

**Listing 4.4** setDatabaseReaders method

```
public void setDatabaseReaders(Pair... args) {
    String databaseFolderPath = MyConfiguration.getDir(FoldersEnum.DATABASE_DIR);

    for (Pair a : args) {
        DatabaseReaderEnum key = (DatabaseReaderEnum)a.getKey();
        String databaseFileName = (String)a.getValue();

        switch (key) {
            case AERODYNAMIC:
                _aerodynamicDatabaseReader =
                    new AerodynamicDatabaseReader(
                        databaseFolderPath,
                        databaseFileName);
                listDatabaseReaders.add(_aerodynamicDatabaseReader);
                break;

            case HIGHLIFT:
                _highLiftDatabaseReader =
                    new HighLiftDatabaseReader(
                        databaseFolderPath,
                        databaseFileName);
                listDatabaseReaders.add(_highLiftDatabaseReader);
                break;
        }
    }
}
```

#### 4.4.4 Developer's guide

In order to execute some analysis in JPAD it is necessary, first of all, to define an analysis object in the Test class. The method `createDefaultAircraft` creates a new aircraft and the object that composes it. This method also populates the data of aircraft with default value corresponding to ATR-72 or Boieng 747\_100B. Moreover the method `createDefaultAircraft` calls another method in Aircraft class: `initialize` that initializes the objects of the classes that perform calculations.

The purpose of this structure is to have only a way to assign the databases at an aircraft. Inasmuch as the wing is always present, the chosen strategy is to assign the database to the aerodynamic manager of the wing.

In order to bring to use the database also for the aircraft calculation, it is assigned at the aerodynamic manager of the aircraft in the method called `doAnalysis`.

At the same time `LSAerodynamicManager` sets itself as aerodynamic in the wing object.

So it is possible to call the database using equally the following codes:

- `theWingObject.getAerodynamics.get_Database;`
- `theAircraftObject.get_theAerodynamic.get_Database;`
- `theLSManagerObject.get_Database;`
- `theACManagerObject.get_Database;`

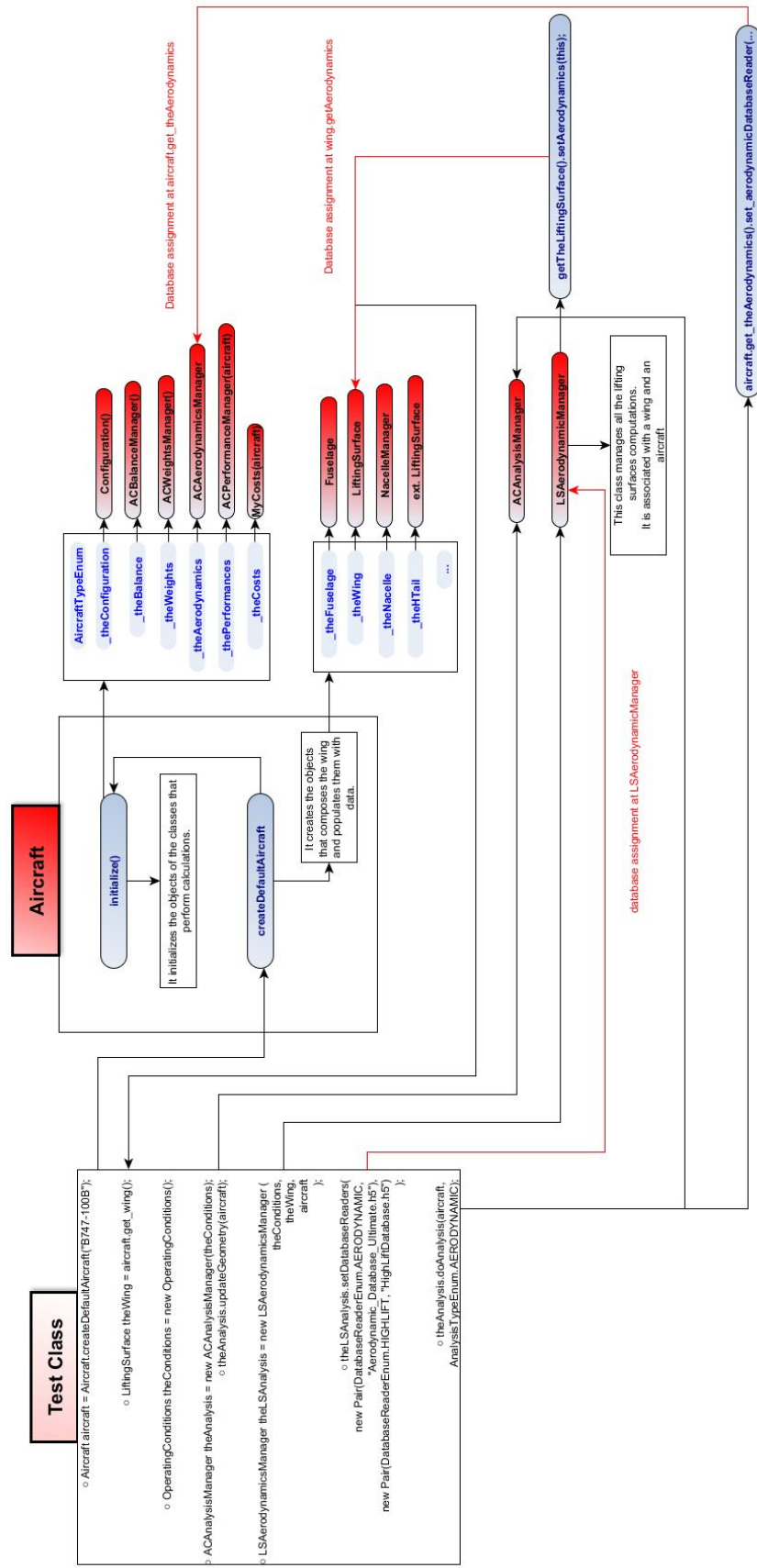


Figure 4.6: Flow chart of database assignment.

# Part III

## Functionality Overview

# Chapter 5

---

## Lift characteristics of a wing

---



# Chapter 6

---

## Drag characteristics of a wing

---

# Chapter **7**

---

## Longitudinal stability of an aircraft

---

# Chapter 8

---

## Minor Works

---

# Appendices

---

## Java Reference

---

### A.1 Inheritance, Overriding and Polymorphism

An extensive usage of inheritance, overriding and polymorphism has been made throughout the application.

Java inheritance can be defined as the process where one object acquires the properties of another. With the use of inheritance the information is made manageable in a hierarchical order. Inheritance has been used several times to define superclass-subclass relationship in order to ease the management of the instances of the subclasses and to group together a set of properties that each subclass must have. Some of the methods defined in the superclass are overridden by the subclasses to execute the proper action accordingly to the run-time object type.

Polymorphism is the ability of an object to take on many forms. The most common use of polymorphism in OOP occurs when a parent class reference is used to refer to a child class object. Any Java object that can pass more than one IS-A test is considered to be polymorphic. In Java, all Java objects are polymorphic since any object will pass the IS-A test for their own type and for the class Object. Polymorphism has been exploited in the design of methods which had to perform an action that could be valid for more than one object type. In such a case, the method signature contains a parameter which is of the superclass type; when invoking the method elsewhere in the application with a subclass instance as an argument at compile time, the compiler uses the method in the superclass to validate the statement in the "client" method. At run time, however, the JVM invokes the method which is defined in the subclass which overrides the superclass method. This behaviour is referred to as virtual method invocation, and the methods are referred to as virtual methods.

In this way it was possible to define a single method which was suitable for several objects types instead of defining a different method for each component or using the instanceof operator.

### A.2 Packages

A Java package is a mechanism for organizing Java classes into namespaces, providing modular programming. Java packages can be stored in compressed files called JAR files, allowing classes to be downloaded faster as groups rather than individually.

Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.

A package provides a unique namespace for the types it contains. In general, a namespace is a container for a set of identifiers (also known as symbols, names). Namespaces provide a level of direction to specific identifiers, thus making it possible to distinguish between identifiers with the same exact name. For example, a surname could be thought of as a namespace that makes it possible to distinguish people who have the same given name. In computer programming, namespaces are typically employed for the purpose of grouping symbols and identifiers around a particular functionality.

## A.3 Documentation

The doc comments, which are a particular type of comments provided by Java, are automatically recognized by the most popular IDE programs; this helps the developer to quickly get an idea of the actions performed by each method, the parameter which has to be passed to it and the what it returns to the user.



---

## Glossary

---

**Aircraft Construction Reference Frame** The reference frame which has its origin in the fuselage forwardmost point, the x-axis pointing from the nose to the tail, the y-axis from fuselage plane of symmetry to the right wing (from the pilot's point of view) and the z-axis from pilot's feet to pilot's head.

**client code** the code where the code in question will be effectively exploited.

**Graphical User Interface** In computing, a Graphical User Interface is a type of interface that allows users to interact with electronic devices through graphical icons and visual indicators such as secondary notation, as opposed to text-based interfaces, typed command labels or text navigation.

**List** The `java.util.List` interface is a subtype of the `java.util.Collection` interface. It represents an ordered list of objects, meaning you can access the elements of a List in a specific order, and by an index too. You can also add the same element more than once to a List.

**Map** The `java.util.Map` interface represents a mapping between a key and a value. The Map interface is not a subtype of the Collection interface. Therefore it behaves a bit different from the rest of the collection types.

**parsing** Parsing or syntactic analysis is the process of analysing a string of symbols, either in natural language or in computer languages, conforming to the rules of a formal grammar.

**reflection** In computer science, reflection is the ability of a computer program to examine (see type introspection) and modify the structure and behavior (specifically the values, meta-data, properties and functions) of the program at runtime.

**serialization** In computer science, in the context of data storage, serialization is the process of translating data structures or object state into a format that can be stored (for example, in a file or memory buffer, or transmitted across a network connection link) and reconstructed later in the same or another computer environment.

**Table** a collection that associates an ordered pair of keys, called a row key and a column key, with a single value. A table may be sparse, with only a small fraction of row key / column key pairs possessing a corresponding value.

**Unified Modeling Language** The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system.



**user developer** the term refers to the developer which will use a method without being interested in how the method performs the required action. This is the case of a utility method: the developer is the one who writes the method, while the user developer is who uses that method to accomplish some action which requires the functionality provided by the utility method. It has to be noticed that the user developer and the developer can be the same person.

**wrapper function** A wrapper function is a subroutine in a software library or a computer program whose main purpose is to call a second subroutine or a system call with little or no additional computation..

---

## Acronyms

---

**ACRF** Aircraft Construction Reference Frame.

**AIAA** American Institute of Aeronautics and Astronautics.

**BRF** Body Reference Frame.

**FAA** Federal Aviation Administration.

**GC** Gravity Center.

**GNC** Guidance Navigation and Control.

**GUI** Graphical User Interface.

**JPAD** Java Aircraft Design.

**MAC** Mean Aerodynamic Chord.

**MAPE** Mean Absolute Percentage Error.

**MLW** Maximum Landing Weight.

**MSL** Mean Sea Level.

**MTOW** Maximum Take Off Weight.

**MZFW** Maximum Zero Fuel Weight.

**UML** Unified Modeling Language.

---

## List of symbols

---

$( )_{\mathbf{H}}$  quantity related to the horizontal tail.

$( )_{\mathbf{LG}}$  quantity related to the landing gear.

$( )_{\mathbf{N}}$  quantity related to the nacelle.

$( )_{\mathbf{S}}$  quantity related to systems.

$( )_{\mathbf{V}}$  quantity related to the vertical tail.

$( )_{\mathbf{F}}$  quantity related to the fuselage.

$( )_{\mathbf{WF}}$  quantity related to the wing-fuselage configuration.

$D$  aerodynamic drag.

$\mathbf{CG}$  Center of Gravity.

$\vec{g}$  gravitational acceleration.

$i_{\mathbf{H}}$  the angle between the horizontal tail root chord and the ACRF x-axis.

$i_{\mathbf{W}}$  the angle between the wing root chord and the ACRF x-axis.

$L$  aerodynamic lift.

$T$  Thrust.

$M$  Mach number.

$\mathcal{AR}$  aspect ratio.

$c$  chord.

$d$  diameter.

$l$  length.

$m$  mass, in kg or lb.

$n_{\mathbf{lim}}$  limit load factor.

$n_{\mathbf{ult}}$  ultimate load factor.

$b$  span.

$S$  surface.

$\Lambda$  sweep.

$\lambda$  taper ratio.

$t$  thickness.

$V$  scalar velocity.

$W$  weight, in Newtons.

$m_W$  wing mass.

$q$  dynamic pressure.

$Re$  Reynolds number (evaluated with respect to  $\bar{c}$ ).

$\alpha_W$  angolo d'attacco riferito alla corda di radice dell'ala.

$\beta$  sideslip angle.

$\rho$  air density.

---

## Bibliography

---

- [1] I. H. Abbott and A. E. von Doenhoff. *Theory of Airfoil Sections*. Dover, 1959.
- [2] Joe Walnes et al. *XStream*. URL: <http://xstream.codehaus.org/>.
- [3] Guillaume Allon. *occjava*. URL: <http://jcae.sourceforge.net/occjava.html>.
- [4] *ATR 72. Aircraft description*. URL: <http://www.atraircraft.com/products/list.html>.
- [5] *ATR 72. Aircraft description*. URL: [https://en.wikipedia.org/wiki/ATR\\_72](https://en.wikipedia.org/wiki/ATR_72).
- [6] Various Authors. *ESDU-76003. Geometrical properties of cranked and straight tapered wing planforms*. London, 2001.
- [7] Various Authors. *eXtensible Markup Language (XML)*. URL: <https://en.wikipedia.org/wiki/XML>.
- [8] Various Authors. *Guava*. URL: <https://github.com/google/guava>.
- [9] Various Authors. *JFace*. URL: <https://wiki.eclipse.org/JFace>.
- [10] Various Authors. *Model View Controller*. URL: <https://en.wikipedia.org/wiki/Model%20%28%80%29%93view%20%28%80%29%93controller>.
- [11] Various Authors. *The Standard Widget Toolkit*. URL: <https://www.eclipse.org/swt/>.
- [12] Various Authors. *Window Builder*. URL: <https://eclipse.org/windowbuilder/>.
- [13] Azeckoski. *ReflectUtils*. URL: <https://code.google.com/p/reflectutils/>.
- [14] A. et al. Bilgin. *Graphviz*. URL: <http://www.graphviz.org/>.
- [15] J. A. Jr Blackwell. *A Finite-Step Method for Calculation of Theoretical Load Distributions for Arbitrary Lifting-Surface Arrangements at Subsonic Speeds*. Washington, D.C., 1969.
- [16] *Boeing 747. Aircraft description*. URL: <http://www.boeing.com>.
- [17] *Boeing 747. Aircraft description*. URL: [https://en.wikipedia.org/wiki/Boeing\\_747](https://en.wikipedia.org/wiki/Boeing_747).
- [18] M. Calcara. *Elementi di dinamica del velivolo*. Napoli: Edizioni CUEN, 1988. ISBN: 9788871460062. URL: <http://books.google.it/books?id=HqBoAQAAAJ>.
- [19] Jean-Marie Dautelle. *Jscience*. URL: <http://jscience.org/>.
- [20] M. Drela and H. Youngren. *XFoils, Subsonic Airfoil Development System*. 2008. URL: <http://web.mit.edu/drela/Public/web/xfoils/>.

- [21] The Apache Software Foundation. *Apache Commons Math*. URL: <http://commons.apache.org/proper/commons-math/>.
- [22] The Apache Software Foundation. *Apache POI*. URL: <http://poi.apache.org/>.
- [23] The Apache Software Foundation. *Apache projects*. URL: <http://www.apache.org/>.
- [24] «Full-configuration drag estimation». In: *Journal of Aircraft* 47 (4) (2010).
- [25] D. Gambardella. «Development of a Java-Based Framework for Aircraft Preliminary Design». Master thesis. University of Naples "Federico II", 2014.
- [26] D. et al Gilbert. *JFreeChart*. URL: <http://www.jfree.org/jfreechart/>.
- [27] R.R. Gilruth and M.D. White. *NACA TR 711 - Analysis And Prediction Of Longitudinal Stability of Airplanes*.
- [28] The HDF Group. *HDF*. URL: <http://www.hdfgroup.org/HDF5/>.
- [29] M. Hepperle. *Applet JavaFoil*. 2006. URL: <http://www.mh-aerotools.de/airfoils/javafoil.htm>.
- [30] W.F. Hilton. *High-speed Aerodynamics*. Longmans, Green, 1951. URL: [https://books.google.it/books?id=%5C\\_RRFAAAAMAAJ](https://books.google.it/books?id=%5C_RRFAAAAMAAJ).
- [31] D. E. Hoak et al. *The USAF Stability and Control DATCOM*. Tech. rep. (Revised 1978). 1960.
- [32] D. Howe. *Aircraft conceptual design synthesis*. Aerospace Series. Professional Engineering Publishing, 2000. ISBN: 9781860583018. URL: <https://books.google.it/books?id=QJZTAAAMAAJ>.
- [33] D. Howe. *Aircraft Loading and Structural Layout*. AIAA Education Series, 2004.
- [34] J. Huwaldt. *1976 Standard Atmosphere Program*. URL: <http://thehuwaldtfamily.org/java/Applications/StdAtmosphere/StdAtmosphere.html>.
- [35] E. N. Jacobs and K. E. Ward. *Interference of wing and fuselage from tests of 209 combinations in the NACA variable-density tunnel*. Tech. rep. 1936.
- [36] L.R. Jenkinson, D. Rhodes, and P. Simpkin. *Civil jet aircraft design*. AIAA education series. Arnold, 1999. ISBN: 9780340741528. URL: <https://books.google.it/books?id=6tMeAQAAIAAJ>.
- [37] K. Kawaguchi. *Args4j*. URL: <http://args4j.kohsuke.org/>.
- [38] I. Kroo and R. Shevell. *Aircraft Design: Synthesis and Analysis*. 1999.
- [39] S. Ledru and C. Denizet. *JLatexMath*. URL: <http://forge.scilab.org/index.php/p/jlatexmath/>.
- [40] W.H. Mason. «Analytic Models for Technology Integration in Aircraft Design». In: AIAA/AHS/ASEE (1990).
- [41] B. W. McCormick. *Aerodynamics, Aeronautics, and Flight Mechanics*. John Wiley & Sons, 1979.
- [42] H. Multhopp. *Aerodynamics of Fuselage*. 1942.
- [43] M. M. Munk. *Aerodynamic Forces of Airship Hulls*. Tech. rep. 1924.
- [44] Marcello R. Napolitano. *Aircraft Dynamics: From Modeling to Simulation*. John Wiley, 2012. ISBN: 9780470626672. URL: <http://www.amazon.com/Aircraft-Dynamics-From-Modeling-Simulation/dp/0470626674>.

- [45] R. Nelson. *Flight Stability and Automatic Control*. McGraw-Hill, 1989.
- [46] R.C. Nelson. *Flight Stability and Automatic Control*. Aerospace Science & Technology. McGraw-Hill International Editions, 1998. ISBN: 9780071158381. URL: <https://books.google.it/books?id=Uzs8PgAACAAJ>.
- [47] L.M. Nicolai and G. Carichner. *Fundamentals of Aircraft and Airship Design*. AIAA education series v. 1. American Institute of Aeronautics and Astronautics, 2010. ISBN: 9781600867514. URL: <https://books.google.it/books?id=qA0oAQAAAMAAJ>.
- [48] M. Nita and D. Scholz. «Estimating Oswald Factor From Basic Aircraft Geometrical Parameters». In: *none* (2012).
- [49] E. Obert. *Aerodynamic Design of Transport Aircraft*. IOS Press, 2009. ISBN: 9781607504078. URL: <https://books.google.it/books?id=axPvAgAAQBAJ>.
- [50] C. D. Perkins and R. E. Hage. *Airplane Performance Stability and Control*. New York: John Wiley & Sons, 1949. ISBN: 9780471680468. URL: <http://books.google.it/books?id=DHxTAAAMAAJ>.
- [51] W. F. Phillips. «Lifting-Line Analysis for Twisted Wings and Washout-Optimized Wings». In: *Journal of Aircraft* Vol. 41.No. 1 (Jan.–Feb. 2004).
- [52] D.P. Raymer. *Aircraft Design: A Conceptual Approach*. 1992.
- [53] D.P. Raymer. *Aircraft Design / RDS-Student: A Conceptual Approach*. AIAA Education Series. Amer Inst of Aeronautics &, 2013. ISBN: 9781600869211. URL: <https://books.google.it/books?id=bvx1lgEACAAJ>.
- [54] J. Roskam. *Airplane Design*. Airplane Design pt. 5. DARcorporation, 1999. ISBN: 9781884885501. URL: <https://books.google.it/books?id=mMU47Ld7yQkC>.
- [55] J. Roskam. *Airplane Design*. Airplane Design pt. 8. DARcorporation, 2002. ISBN: 9781884885556. URL: <https://books.google.it/books?id=GIHhFkd829cC>.
- [56] J. Roskam. *Airplane Flight Dynamics and Automatic Flight Controls*. DARcorporation, 2001.
- [57] J. Roskam. *Airplane Flight Dynamics and Automatic Flight Controls*. Airplane Flight Dynamics and Automatic Flight Controls. Darcorporation, 2003. ISBN: 9781884885181. URL: <http://books.google.it/books?id=pWUAzy87zLYC>.
- [58] M.H. Sadraey. *Aircraft Design: A Systems Engineering Approach*. Aerospace Series. Wiley, 2012. ISBN: 9781118352809. URL: <https://books.google.it/books?id=VT-Tc3Tx5aEC>.
- [59] OPEN CASCADE S.A.S. *OpenCASCADE*. URL: <http://www.opencascade.org/>.
- [60] P.M. Sforza. *Commercial Airplane Design Principles*. Elsevier Science, 2014. ISBN: 9780124199774. URL: <https://books.google.it/books?id=knhHAqAAQBAJ>.
- [61] E. Torenbeek. *Advanced Aircraft Design: Conceptual Design, Technology and Optimization of Subsonic Civil Airplanes*. Aerospace Series. Wiley, 2013. ISBN: 9781118568095. URL: <https://books.google.it/books?id=C0gUyoYewhoC>.
- [62] E. Torenbeek. *Synthesis of Subsonic Aircraft Design*. 1976.

- [63] E. Torenbeek. *Synthesis of Subsonic Airplane Design: An Introduction to the Preliminary Design of Subsonic General Aviation and Transport Aircraft, with Emphasis on Layout, Aerodynamic Design, Propulsion and Performance*. Springer, 1982. ISBN: 9789024727247. URL: [https://books.google.it/books?id=NG2%5C\\_qiSjmMEC](https://books.google.it/books?id=NG2%5C_qiSjmMEC).
- [64] Dimitri Van Heesch. *Doxygen*. 1997-2014. URL: [www.doxygen.org](http://www.doxygen.org).
- [65] S. R. Vukelich and J. E. Williams. *The USAF Stability and Control Digital Database*. AFFDL-TR-79-3032. Updated by Public Domain Aeronautical Software 1999. Apr. 1979. Volume I.