Java Decisioning Program

Benjamin Roberts

Level 4 Software Development

Contents

[Introduction & Requirements 3](#_Toc168238531)

[Project Description 3](#_Toc168238532)

[How this was decided 3](#_Toc168238533)

[Acceptance Criteria/Project Outcomes 3](#_Toc168238534)

[Stakeholders Involved 4](#_Toc168238535)

[Development 5](#_Toc168238536)

[User Input 5](#_Toc168238537)

[Data Model 7](#_Toc168238538)

[CRUD - Database Connection 9](#_Toc168238539)

[Access Management 9](#_Toc168238540)

[Handing Azure firewall 9](#_Toc168238541)

[Credit Policy built 10](#_Toc168238542)

[Scorecards 10](#_Toc168238543)

[Policy Rules 18](#_Toc168238544)

[Testing the program 19](#_Toc168238545)

[JUnit 19](#_Toc168238546)

[End to End 20](#_Toc168238547)

[Final Code Repository 20](#_Toc168238548)

[Conclusions 20](#_Toc168238549)

[Project Outcome 20](#_Toc168238550)

[Personal Reflections 20](#_Toc168238551)

[Future Additions 20](#_Toc168238552)

[References 21](#_Toc168238553)

[Probability of Default 22](#_Toc168238554)

[Appendix 23](#_Toc168238555)

[Resources: 23](#_Toc168238556)

[Communications 24](#_Toc168238557)

[Project Brief & Checklist 25](#_Toc168238558)

[Learning Criteria: 26](#_Toc168238559)

# Introduction & Requirements

## Project Description

Create a Java program that implements Credit Decisioning logic at an introductory level. This must include:

1. Some ability to capture relevant applicant information required to make a credit decision and enter this information into the program
2. Execute credit decisioning policy rules that conditionally reject an application if set criteria is met
3. Connect to an external database to retrieve more data to use for decisioning
4. Use additional data gathered from this database to execute Scorecards in a champion/challenger approach (requiring at least 2 scorecards to be created)
5. Return a final decision and an offer to the user depending on the decisioning logic
6. A record of the program run is then saved into an external database (for monitoring and further credit decisioning use)

## How this was decided

The team I work in is primarily responsible for developing and maintaining a Credit Decisioning software platform. The software is provided by a 3rd party and provides a low-code graphic user interface for developing business logic into a callable Web service without the need for specialised IT support.

The Java elements of the Module 1 course demonstrated an opportunity to apply similar business logic into a more technical language, in a role relevant manner.

## Acceptance Criteria/Project Outcomes

1. Final program must have an interface for a user to input application data
2. Application data input must include:
   1. A Match Id of some kind to look up in the external database
   2. Residential Status
   3. Employment Status
3. A database connection is made to allow CRUD operations for logging activity made in the program
4. A database connection is made to allow a lookup to be made based on the data received in the program
5. Multiple credit decisioning policy rules are built and executed in the program flow
   1. Policy rules are grouped in a modular way to breakdown testing
   2. A mix of the application input data and data retrieved from the external database is used
6. Multiple credit decisioning scorecards are built and executed
   1. A credit offer is made in a champion/challenger style i.e. only 1 is used as the basis for the offer despite both being executed

# Stakeholders Involved

To develop the project, I consulted a number of stakeholders for support. This ranged from:

* Confirmation the project was applicable to the workplace
* Advice on setting up resources to use for the project e.g. SQL Databases for CRUD operations

A summary of the input from each stakeholder is recorded below:

|  |  |  |
| --- | --- | --- |
| **Stakeholder** | **Type** | **Input** |
| Credit Risk | Internal | * Sign-off for project with Line Manager * Consult for specific business logic that could be applied to the project & would be appropriate to share |
| IT Delivery Managers | Internal | * Consulted for advice on how to host the application, specifically for the database link for CRUD operations |
| Asset Management | Internal | * Arranged the installation of all software and licenses required for this project, including:   + Eclipse IDE   + JDBC Driver   + Visual Studio 2022 subscription with Microsoft Azure credits |
| Data Delivery | Internal | Consulted for:   * advice on setting up an Azure SQL Database user * importing data into a database that could be used for lookup operations |
| Model Management | Internal |  |
| QA Digital Learning Consultant | External | * Consulted my DLC for information on Maven Projects to build |
| QA Additional Learning Support | External |  |

Where possible, I have also included written communications (e.g. Emails, Bud Messages) in the `[communications](#_Communications)` appendices.

# Development

## User Input

The Java Scanner class was used to prompt the user for input. Defined below is a class called “UserInput” which has been written to group all methods related to capturing user input together:

A screenshot of a computer program

Description automatically generated

**Figure**: Code used to define the “UserInput” class, importing the necessary Scanner class & defining

Upon running the program, the user will be asked a series of questions:

1. Their Forename
2. Surname
3. The ResidentialStatus and their EmploymentStatus, presented as a menu. An example of the code for this is annotated below:

A screen shot of a computer program

Description automatically generated

* “println()” functions print out a menu of options to the console for the user
* A custom defined “getIntInput” method calls the Java Scanner & writes out another prompt to the console with instructions to the user
* A “while” loop traps the user in this loop until they provide a valid option
* The “try-catch” block is included to capture invalid characters e.g. string values when expecting a value of 1 to 6
* The “loopCount” increment logic is included as a failsafe for ending the while loop in case of too many bad inputs and/or preventing an infinite loop

## Data Model

One issue I ran into early on was how to logically structure the data used within the program. When working with Credit Decisioning systems in the workplace, we use a Data Model to define what data the application has access to and how it is stored.

I chose to apply a similar approach to this program, by defining an “AppData” class:

A screenshot of a computer

Description automatically generated

**Figure: the “AppData” class written to act as a Data Model**

Doing this gave these advantages:

1. All application data is logically grouped
2. A single data object can be logged for review
3. Java Methods could be written to accept this model, allowing some flexibility around Java’s strongly typed data constraints when writing methods

Once the program completes it’s run, this Data model is populated with all the data used during the run. This allows the data to be logged easily for review. To do this, I made use of the GSON library to get a JSON representation of this:

**A screen shot of a computer

Description automatically generated**

**Figure: Importing and using the GSON library to create a JSON Representation to log to the console**

An abridged example output of this is below:

{

"appl": {

"forename": "T",

"surname": "T",

"employmentStatus": "FullTime",

"residentialStatus": "HomeOwner"

},

"decData": {

"pk": 4874,

"E1B07": "0",

"E1B08": "0",

"E1B09": 2,

"TRD\_A\_13": -3,

"E1A09": 0,

"E5S051": 777,

"NDSPCII": -4,

"TRD\_A\_06": 2,

"E1B13": 0,

"E1B01": 0,

"TRD\_STL\_14": -2,

"TRD\_STL\_19": -2

},

"scorecardRngResult": 77,

"strategyPath": "CHALLANGER",

"scorecards": [

{

"scorecardName": "ChampionScorecard",

"score": 733,

"index": 0,

"probOfDefault": 0.0024726231566347745,

"stringCharacteristics": [

{

"characteristicValue": "0",

"name": "E1B07",

"bandingScore": 27

},

{

"characteristicValue": "HomeOwner",

"name": "ResidentialStatus",

"bandingScore": 40

}

],

"numberCharacteristics": [

{

"characteristicValue": 2,

"name": "E1B09",

"bandingScore": 10

},

{

"characteristicValue": -3,

"name": "TRD-A-13",

"bandingScore": 40

},

{

"characteristicValue": 0,

"name": "E1A09",

"bandingScore": 46

},

{

"characteristicValue": -2,

"name": "TRDSTL14",

"bandingScore": -30

}

]

},

{

"scorecardName": "ChallangerScorecard",

"score": 691,

"index": 0,

"probOfDefault": 0.01798620996209156,

"stringCharacteristics": [

{

"characteristicValue": "FullTime",

"name": "EmploymentStatus",

"bandingScore": 45

}

],

"numberCharacteristics": [

{

"characteristicValue": -4,

"name": "NDSPCII",

"bandingScore": -40

},

{

"characteristicValue": 2,

"name": "TRD-A-06",

"bandingScore": 14

},

{

"characteristicValue": 0,

"name": "E1B13",

"bandingScore": 26

},

{

"characteristicValue": 0,

"name": "E1B01",

"bandingScore": 34

},

{

"characteristicValue": -2,

"name": "TRD-STL-19",

"bandingScore": 32

}

]

}

],

"flags": [

{

"policyCode": "D\_001",

"ruleOutcome": "PASS",

"decisionType": "DECLINE"

},

{

"policyCode": "D\_002",

"ruleOutcome": "PASS",

"decisionType": "DECLINE"

},

{

"policyCode": "D\_003",

"ruleOutcome": "PASS",

"decisionType": "DECLINE"

},

{

"policyCode": "A\_001",

"ruleOutcome": "PASS",

"decisionType": "ACCEPT"

}

]

}

**Figure: An example of the GSON output, returning a JSON Representation of the Java Program's data model**

## CRUD - Database Connection

### Access Management

### Handing Azure firewall

A screenshot of a computer

Description automatically generated

**Figure: A screenshot showing a Firewall rule in place on my database to only allow access from my network (with IP Addresses redacted)**

## Credit Policy built

The below section details the Credit Decisioning Logic built into the program.

### Scorecards

Credit Scorecards are a tool in credit risk management used by Credit Risk Management teams to model/predict the likelihood of default for a given subject borrower (i.e. an applicant/customer) over a period (Huang and Scott, 2017). They are used for evaluating the creditworthiness of a customer, making them critical part of the Credit Decisioning process. This made them a natural choice to build in this project.

Scorecards from a program point of view are made up of:

* Characteristics which receive a score based on their value
* A final calculated score based on the sum of the characteristic scores + a starting score
* A “Score Index” used to group ranges of scores together. Typically translates into groups the business can use to simply communicate which applicants are more risky
* A “Probability of Default”, which translates the raw score into a percentage chance the applicant would default on a credit offer i.e. they are unable to pay

This project implements 2 scorecards in a champion/challenger fashion. This is an approach that deploys multiple competing strategies into a production environment and monitoring which provides the best performance (FICO, 2020). Using this is an effective demonstration of using Object-Oriented programming to build Credit Decisioning logic.

Note that both scorecards are not based on any actual statistical model. They are only for demonstrating how the concept can be implemented in an Object-Oriented program.

The Characteristics form the core of any scorecard. Given that each scorecard is unique in what variables it uses and how it assesses each variable, I chose to use classes containing static methods to implement these. This is shown in the figure below:

A screen shot of a computer

Description automatically generated

**Figure: Java Classes for both Scorecards.**

#### Decide between Champion and Challenger

For a Champion/Challenger strategy, the goal is to send a percentage of applicants down the challenger path and compare the outcomes to the champion. To start, most of the volume would be directed to the known good model (i.e. the champion). This allows the business to test the challenger model with a limited volume to analyse if better business outcomes are given by the challenger.

To apply this to my program, I added logic into my AppData model constructor to decide which strategy path the program should follow. This logic is below:

A screen shot of a computer program

Description automatically generated

**Figure: View of the AppData Class constructor, which the Champion/Challenger split**

Defining this logic in the constructor of the Data Model allows me to easily ensure this split is made early in the program. It is implemented as:

* 60% of the users will use the Champion Path
* 40% use the Challanger Path

The “Random” Java class is used to generate a random number to enforce this split.

#### Characteristics

These scorecards consist of a starting/base score plus a series of characteristics, variables with values that represent a statistical insight into the applicant/customer e.g. the E1A09 characteristic in the Champion scorecard count how many delinquent (i.e. in arrears) account they have present on their account (Experian, 2024). Specific values in that variable get assigned Banding Scores (or Points) which represent if they are a positive or negative indication of the applicant’s credit worthiness.

Once calculated, all the Banding scores are added together, along with an initial starting/base score to calculate a final score for the applicant, which represents their credit worthiness. A financial institution can use this as a basis for deciding a credit offer i.e. do they offer them a product and what terms they will offer if they do.

Below are specifications of both the Champion and Challenger scorecards built into the program, documented in a style typical in the workplace. The columns mean:

* **Characteristic** = the name of the variable used. Can reference ether data the applicant enters on an application form or data available on their Credit records accessed via Credit Bureaus like Experian
* **Data Type –** documents the type of values the characteristic can hold. Useful for guiding developeers in how to handle the values within
* **Min Value & Max Value –** These relate to the **Points** column & defines the ranges used to set a specific Banding Score/Points. For String values, this is a 1-to-1 mapping for value to points
* **Points –** The actual Banding score assigned to the characteristic, which is then added to the overall score on the scorecard

#### Champion Specification

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Characteristic** | **Data Type** | **Min Value** | **Max Value** | **Points** | **Notes** |
| Base Score | Numeric | [default] | | 600 | This is a static value that all instances of this scorecard will start from. |
| E1B07 | String | T | | 0 | External data provided by Experian’s Credit Record API: DelphiSelect  Defined as: ***Worst Status in the last 6 months of all Active CAIS accounts (SP)***   (Experian, 2024). |
| N | | 0 |
| D | | 0 |
| U | | 0 |
| 0 | | 27 |
| 1 | | 14 |
| 2 | | -10 |
| 3 | | -23 |
| 4 | | -23 |
| 5 | | -23 |
| 6 | | -23 |
| 8 | | -50 |
| [default] | | -50 |
| E1B09 | Numeric | -1 | 0 | -20 | External data provided by Experian’s Credit Record API: DelphiSelect  Defined as***: Number of Active CAIS Accounts (SP)*** (Experian, 2024). |
| 1 | 2 | 10 |
| 3 | 4 | 20 |
| 5 | 6 | 30 |
| 7 | max | 18 |
| [default] | | -20 |
| TRD-A-13 | Numeric | -3 | -3 | 40 | External data provided by Experian’s Credit Record API: DelphiSelect  Defined as***: Number of Months Since Most Recent Arrears on all Accounts allowed*** (Experian, 2024). |
| -2 | -2 | -20 |
| -1 | 0 | -20 |
| 1 | 1 | 10 |
| 2 | 2 | 5 |
| 3 | max | -30 |
| [default] | | -30 |
| E1A09 | Numeric | -1 | -1 | 0 | External data provided by Experian’s Credit Record API: DelphiSelect  Defined as: ***Number of Delinquent CAIS Accounts, excluding Mail Order accounts*** (Experian, 2024). |
| 0 | 0 | 46 |
| 1 | 2 | 24 |
| 3 | max | -26 |
| [default] | | -26 |
| TRD-STL-14 | Numeric | -2 | -2 | 26 | External data provided by Experian’s Credit Record API: DelphiSelect  Defined as: ***TSMR Active Short Term Loans account opened (Active CAIS)*** (Experian, 2024). |
| -1 | -1 | 0 |
| 0 | 0 | 26 |
| 1 | 6 | -24 |
| 7 | 18 | -15 |
| 19 | 36 | 6 |
| 36 | max | 15 |
| [default] | | -24 |
| ResidentialStatus | String | HomeOwner | | 40 | This data point is one of the User Input questions.   The status indicates the type of accommodation someone lives in. |
| PrivateTenantFurnished | | 26 |
| PrivateTenantUnfurnished | | 34 |
| CouncilTenant | | 10 |
| Cohabiting | | 34 |
| LivingWithParents | | 10 |
| [default] | | 10 |

#### Challenger Specification

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Characteristic** | **Data Type** | **MIn Value** | **Max Value** | **Points** | **Notes** |
| Base | Numeric | [default] | | 580 | This is a static value that all instances of this scorecard will start from. |
| NDSPCII | Numeric | -3 | -1 | 0 | External data provided by Experian’s Credit Record API: DelphiSelect  Defined as: ***Consumer Indebtedness Index (SP)*** (Experian, 2024). This is an index that represents how in debt a person is. Higher values indicate the subject is using more of their available credit. |
| 1 | 10 | 45 |
| 11 | 20 | 23 |
| 21 | 25 | 14 |
| 26 | 40 | 5 |
| 41 | 50 | -20 |
| 51 | max | -40 |
| default | | -40 |
| TRD-A-06 | Numeric | -2 | -2 | 0 | External data provided by Experian’s Credit Record API: DelphiSelect  Defined as: ***Number of Consistently up-to-date accounts L12m*** (Experian, 2024). |
| -1 | -1 | -10 |
| 0 | 0 | -20 |
| 1 | 1 | 6 |
| 2 | 2 | 14 |
| 3 | max | 21 |
| default | | -20 |
| E1B13 | Numeric | -1 | -1 | -10 | External data provided by Experian’s Credit Record API: DelphiSelect  Defined as: ***Number of CAIS Status 3 or worse, within the last 6 Months for all Active CAIS accounts (SP)*** (Experian, 2024). CAIS Status refers to the arrears position of the account e.g. 3 means 3 missed payments on the account. |
| 0 | 0 | 26 |
| 1 | 1 | -12 |
| 2 | max | -56 |
| default | | -56 |
| E1B01 | Numeric | -1 | -1 | 0 | External data provided by Experian’s Credit Record API: DelphiSelect  Defined as: ***Number of Active CAIS Accounts Opened in the Last 3 Months (SP)*** (Experian, 2024). |
| 0 | 0 | 34 |
| 1 | 1 | 12 |
| 2 | 2 | -11 |
| 3 | max | -33 |
| default | | -33 |
| TRD-STL-19 | Numeric | -2 | -2 | 32 | External data provided by Experian’s Credit Record API: DelphiSelect  Defined as: ***Total Number of Default Short Term Loans accounts (Default CAIS)*** (Experian, 2024). |
| -1 | -1 | 0 |
| 0 | 0 | 28 |
| 1 | 1 | -14 |
| 2 | max | -27 |
| default | | -27 |
| EmploymentStatus | String | FullTime | | 45 | This data point is one of the User Input questions.   The status indicates the type of employment an applicant is in. |
| PartTime | | 24 |
| Student | | 6 |
| SelfEmployed | | 33 |
| Unemployed | | -10 |
| Retired | | 22 |
| default | | -10 |

#### NULL Handling in characteristics

NULL values are technically possible within the External Data variables detailed in the specifications above. In this program the expectation is that those should fall into the “default” bands and get assigned the relevant points.

To handle this in Java, try-catch blocks are used to specifically catch a NullPointerException that would arise from attempting to use these values e.g. the screenshot below shows this applies to the E1B09 characteristic on the Champion scorecard:

A screen shot of a computer program

Description automatically generated

**Figure: NULL handling for a Scorecard characteristic.**

#### Probability of Default formula

The calculated score on a Scorecard is intended to be a representation of how likely a subject is likely to default on any offered credit. In its natural form however this isn’t a clear value to communicate probability.

The Probability of Default formulas allow us to translate this raw score into a percentage value that can be more cleanly communicated to non-technical stakeholders. It makes use of coefficients (called “beta” β) and model variables defined early on in the Scorecard Model Development process to convert the raw score into a percentage value.

An accepted model for calculating this value (based on Logical Regression) is provided below:

|  |  |
| --- | --- |
| Probability of Default formula | **e** = Eular’s Number (i.e. exponent)  **β** = beta (i.e. the model coefficients)  **x** = Model variables defined in the Scorecard Model Development process |

**Figure: Probability of Default formula (Silva et al, 2020).**

Beta i.e**. β** is a static value defined as part of the development process. Doing so requires using historical data on the population we’re modelling for. This data is run through statistical analysis (e.g. logical regression) to estimate a co-efficient to use.

X r

Defining the constants to use in this model is typically done in the model development process. As only mock scorecards are used for this program to demonstrate the scorecards concept, the output of this will have no statistically meaningful value. The focus is solely on the Java implementation.

To apply this concept to my mock scorecards, I have taken an implementation of this from an older Scorecard no longer in use. That implementation used the formula:

**PD = '1/(2^((Score-[basescore] )/[point jump)+1)**

This screenshot below shows this coded in the “Scorecard.java” class:

A screen shot of a computer program

Description automatically generated

Where:

* Math.exp() function provides the Exponent value required for the formula
* baselineScore & pointsJump represent the constants to be uses in the formula

### Policy Rules

Policy rules check the available data on the applicant for specific signs of financial difficulty or any other indication that the applicant may be outside the credit institutions risk appetite.

Policy Rules can be of 1 of these types:

* **Decline** = If the logic is met, no offer of credit given to the applicant
* **Accept** = Valid only if the logic for all instances of the other 2 types are not met. Represents no concerns found on the provided data, an offer of credit can be given

|  |  |  |  |
| --- | --- | --- | --- |
| **Code** | **Rule Name** | **Decision** | **Logic** |
| D\_001 | Worst Arrears status is 2+ payments behind | Decline | IF DecisioningDataRow.E1B08 regex match 2|3|4|5|6|8 THEN FAIL |
| D\_002 | Experian Score below threshold | Decline | IF DecisioningDataRow.E5S051<= 600 THEN FAIL |
| D\_003 | Scorecard | Decline | IF (  decisionPath = “CHAMPION” AND Scorecards.Champion.score < 600  )  OR (  Scorecards.Challenger.score < 580  ) THEN FAIL |
| D\_004 | Unemployed | Decline | If EmploymentStatus == “Unemployed” |
| A\_001 | Accept | Accept | If none of the above Policy rules are hit |

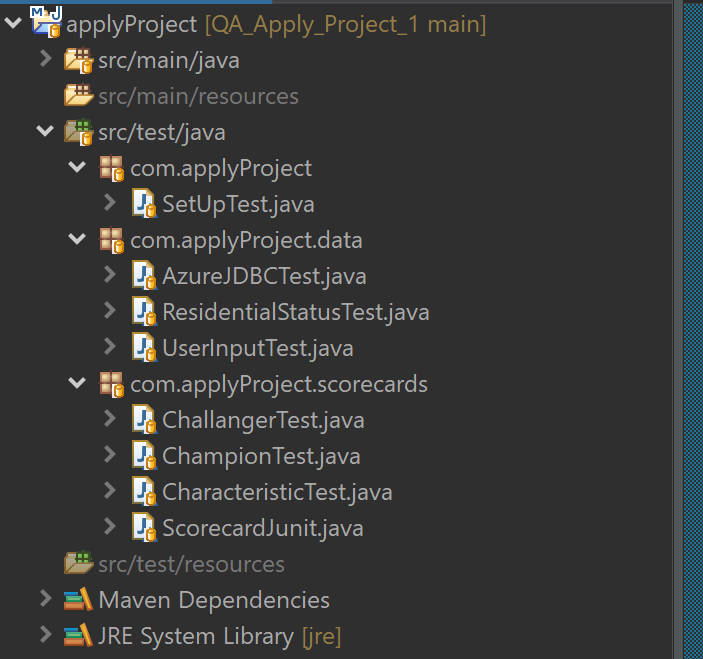
# Testing the program

To be sure my program worked as intended and to the project requirements, testing was necessary. This was approached in 2 ways:

* Unit Tests via Junit 5
* End to end tests via running the program in the Eclipse IDE

## JUnit

When programming logic within the bespoke tool I use in my workspace, Unit Tests are used to ensure components of the code work as expected before a deployment is done. These are small scale tests that can be separated from the main program & re-used when any changes are made, allowing code quality to be maintained (AWS, n.d.). JUnit 5 is a Java package that provides a frame work for writing these test & executing them (TutorialsPoint.com, n.d.).



**Figure: JUnit 5 Test cases written to test individual parts of the program independently**

A screen shot of a computer

Description automatically generated

**Figure: A test case written to test a program set up class that reads properties from an external file**

## End to End

# Final Code Repository

All code developed for this project can be found in the below GitHub Repository:

<https://github.com/btr6566/qam1_java_decisioning_app>

# Conclusions

## Project Outcome

## Personal Reflections

## Future Additions

* Front end interface to collect more data on an applicant e.g:
  + Addresses
  + Income
  + Expenditure
* Submit the collected applicant data to the program as a semi-structured message body e.g. JSON
* Replace the random database lookup with a connection to the Delphi Select Sandbox API

# References

AWS (n.d.). *What is Unit Testing? - Unit Testing Explained - AWS*. [online] Amazon Web Services, Inc. Available at: <https://aws.amazon.com/what-is/unit-testing/>.

Bionic Turtle. (2011). FRM: Logistic distribution maps credit score to probability of default (PD). [Online]. Youtube. Last Updated: 2011. Available at: <https://www.youtube.com/watch?v=HnQ6QjeGMm0> [Accessed 9 May 2024].

Experian. (2024). DelphiSelect API Documentation - for Credit Decisioning data. [Online]. DelphiSelect API Documentation. Last Updated: 2024. Available at: <https://developer.experian.com/products/uk/delphi-select/delphi-select-api-docs> [Accessed 17 May 2024].

FICO. (2020). A Guide to Champion/Challenger Testing Approaches: Overview and Benefits. [Online]. FICO Community. Last Updated: 2020. Available at: <https://community.fico.com/s/blog-post/a5Q2E000000YNqoUAG/fico2227> [Accessed 17 May 2024]

Huang and Scott. (2017). Credit Risk Scorecard Design, Validation and User Acceptance. [Online]. Credit Risk Scorecard Design, Validation and User Acceptance. Last Updated: 2017. Available at: <https://cer.business-school.ed.ac.uk/wp-content/uploads/sites/55/2017/03/Credit-Risk-Scorecard-Design-Huang-and-Scott.pdf> [Accessed 17 May 2024].

QA Apprenticeships. (2024). Module 1 - Java JDBC Slides. [Online]. QA Apprenticeships. Last Updated: 2024. Available at: <https://github.com/broberts-gtw/LVL_4_Software_Dev_Repo/blob/main/Module%201%20-%20Intro%20to%20Software%20Development/Live%20Course%201%20Materials/__Course%20Stretch%20files/Java/presentation/JDBC.pptx> [Accessed 1 May 2024].

Microsoft Azure. (2010). Adding Users to Your SQL Azure Database. [Online]. azure.microsoft.com. Last Updated: June 21, 2010. Available at: <https://azure.microsoft.com/en-us/blog/adding-users-to-your-sql-azure-database/> [Accessed 3 May 2024].

Microsoft. (2023). Northwind Database - TSQL. [Online]. Northwind Database - TSQL. Last Updated: 2023. Available at: <https://github.com/microsoft/sql-server-samples/blob/master/samples/databases/northwind-pubs/instnwn> [Accessed 1 May 2024].

Silva et al. (2020). A logistic regression model for consumer default risk. [Online]. National Library of Medicine. Last Updated: 2020. Available at: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9041570/> [Accessed 9 May 2024].

Stack Overflow. (2008). Can an abstract class have a constructor?. [Online]. Stack Overflow. Last Updated: Updated Dec 30, 2016. Available at: <https://stackoverflow.com/questions/260666/can-an-abstract-class-have-a-constructor> [Accessed 3 May 2024].

Stack Overflow - ArrayList. (2021). Adding objects to an array list -. [Online]. Stack Overflow - ArrayList. Last Updated: 2021. Available at: <https://stackoverflow.com/questions/70406298/adding-objects-to-an-array-list-cannot-invoke-xxx-add-b> [Accessed 14 May 2024].

TutorialsPoint.com. (n.d.). JUnit - Test Framework - Tutorialspoint. [online] Available at: https://www.tutorialspoint.com/junit/junit\_test\_framework.htm.

W3Schools. (Unknown). Java Abstraction. [Online]. W3Schools. Available at: <https://www.w3schools.com/java/java_abstract.asp> [Accessed 3 May 2024].

## Probability of Default

Anderson, R. (2007). The Credit Scoring Toolkit: Theory and Practice for Retail Credit Risk Management and Decision Automation. Oxford University Press.

Mays, E. (2004). Credit Scoring for Risk Managers: The Handbook for Lenders. South-Western Educational Publishing.

Siddiqi, N. (2017). Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring. Wiley.

Thomas, L. C., Edelman, D. B., & Crook, J. N. (2002). Credit Scoring and Its Applications. Society for Industrial and Applied Mathematics.

# Appendix

## Resources:

[Java Interface (w3schools.com)](https://www.w3schools.com/java/java_interface.asp)

[Java Abstraction (w3schools.com)](https://www.w3schools.com/java/java_abstract.asp)

[Java Exceptions (Try...Catch) (w3schools.com)](https://www.w3schools.com/java/java_try_catch.asp)

## Communications

## Project Brief & Checklist

The following list is a reminder and checklist on what you should include in your final submission.

* **Introduction**
  + Description of the project.
  + The approach.
  + Project outcomes.
  + How the KSBs are evidenced through your workplace activity.
  + How the project was assigned to you.
  + The stakeholders you liaised with for this.
  + The key requirements.
  + Other considerations e.g., budget.
  + The way you interacted with various stakeholders.
* **Logic and maths**
  + write good quality code (logic) with sound syntax in at least one language.
  + apply the maths required to be a software developer (e.g. algorithms, logic and data structures).
  + Additionally, make sure you show evidence that you have applied the following:
    - Elements of programming: variables, assignment statements, data types, conditionals, loops, arrays, and input / output
    - Functions: modular programming dividing a program into components that can be independently debugged, maintained, and reused writing at least two reusable functions
    - Algorithms and data structures: classical algorithms for sorting and searching and fundamental data structures
* **Linking data**
  + effectively link code to the database and data sets.
  + link to a range of database types and embed data queries within your code.
  + make a connection to the database.
  + execute CRUD statements on the database.
  + use one-off queries and stored procedures.
  + transform returned data into a format required by the application.

## Learning Criteria:

S7 - Apply structured techniques to problem solving, can debug code and can understand the structure of programmes to identify and resolve Issues

S4 - Test code and analyse results to correct errors found using unit testing

S1 - Create logical and maintainable codes

B10 - Committed to continued professional development

B4 - Works collaboratively with a wide range of people in different roles, internally and externally, with a positive attitude to inclusion and diversity

B7 - Communicates effectively in a variety of situations to both a technical and non- technical audience

B5 - Acts with integrity with respect to ethical, legal and regulatory ensuring the protection of personal data, safety and security

B1 - Works independently and takes responsibility. For example, has a disciplined and responsible approach to risk, and stays motivated and committed when facing challenges

B6 - Shows initiative for solving problems within their own remit, being resourceful when faced with a problem to solve

B8 - Shows curiosity to the business context in which the solution will be used, displaying an inquisitive approach to solving the problem. This includes the curiosity to explore new opportunities, and techniques; the tenacity to improve methods and maximise performance of the solution; and creativity in their approach to solutions

B9 - Demonstrates creativity and tenacity in their approach to solutions and the methods used to come to a solution, for example sees the task through to the end by devising new solutions and despite obstacles and problems along the way