# Assessment Checklist

You are required to implement the program in an object oriented manner. The solution must demonstrate each of the following, shown in the tables below. To help you to ensure that you have covered all the points required in the implementation of the a

Assessment please complete the tables below for each point. Include examples, observations, and justification where appropriate.

## Design requirements:

|  |
| --- |
| **Abstraction, encapsulation and information hiding used where appropriate** |
| Examples: In the program, “Tile” an abstract class as it isn’t directly used at any point. It also has “getters” and “setters” to give the xCoord and yCoord of the Tile via encapsulation. The xCoord and yCoord attributes are set to private thus making the getters and setters the only way to access them. |

|  |
| --- |
| **Inheritance used if appropriate to the solution** |
| Examples: There is numerous examples of inheritance in the program such as “Tile” inheriting “~~Coords~~”, “MoveableTile” inheriting “Tile”. This is to ensure there is a relationship between these classes as they require each other to work and function properly. |

|  |
| --- |
| **Polymorphism used if appropriate to the solution** |
| Examples: The “Tile” class is polymorphic as it inherits an abstract class “GUI” AND an interface “TilePlacement”, this also makes all the subclasses of “Tile” polymorphic as well. |

|  |
| --- |
| **All class-wide variables are private to prevent content coupling class-wide variables are kept to a minimum to ensure a minimum of common coupling** |
| Examples: All the attributes in every class are given a private scope to prevent content coupling, this was done to avoid confusion and overlapping variable names. Class-wide attributes were kept to a minimum, instead focusing on local method variables to avoid common coupling too. |

|  |
| --- |
| **Data coupling is used (parameter passing) in preference to content or common coupling** |
| Examples: There is a use of parameter passing, especially in “MoveableTile” where multiple methods are called that require variables. Most constructors also use parameters when they are called in order to create an object correctly. |

|  |
| --- |
| **Program does not contain a lot of unnecessary data coupling** |
| Examples: When a constructor or method was created, only what variables it required were put up for data coupling/parameter passing to reduce its usage as much as possible |

|  |
| --- |
| **Classes are highly cohesive** |
| Examples: Each class has its own purpose and doesn’t contain methods or attributes that aren’t relevant to its own object. |

## Implementation requirements:

|  |
| --- |
| **A working solution which meets the requirements of the given brief** |
| Comments: The program meets the requirements and utilises all the required methods. |

|  |
| --- |
| **Variables are correctly declared and initialised** |
| Examples: There is no unused variables or improperly initialised ones. Such as the various variable used in the “Level” class |

|  |
| --- |
| **Arithmetic and/or logical operators are used correctly** |
| Examples: Where needed, mathematical operators are correctly used, logic operators in if statements are also correct if a &&AND, ||OR or !NOT operator is needed. |

|  |
| --- |
| **A range of control structures are implemented correctly** |
| Examples: There is control structures in the form of IF statements, Switch Statements, loops, breaks and returns. All implemented correctly. A switch statement is implemented in the “Level” class for example which is used to generate and populate the grid required for gameplay |

|  |
| --- |
| **At least two data structures are implemented correctly** |
| Examples: There is both an Array and an ArrayList implemented in the program. ArrayLists are more commonly used due to their easier useable. But one normal Array also exists. Which consequently is stored in an ArrayList thus creating a multi-dimensional array. |

|  |
| --- |
| **The program contains a minimum of four classes, which contain attributes, methods and a constructor method** |
| Examples: There is 13 classes, 3 of which are abstract. There is also 1 interface. The classes contain attributes, methods and (in the non-abstract classes) constructors. |

|  |
| --- |
| **A minimum of three objects are created from the classes, with appropriate initial attribute values set through the constructor methods** |
| Examples: As mentioned above, 10 classes can create objects via their constructors. Most of the constructors also require parameters passed into them in order to create the objects properly. |

|  |
| --- |
| **The program contains at least one overloaded method (this may be a constructor method)** |
| Examples: The “Floor” class contains an overloaded constructor in order to avoid a null tile being placed if the game is unable to place down a correct tile. This ensures the game can be still playable. |

|  |
| --- |
| **Classes are linked appropriately through association, aggregation or inheritance relationships** |
| Examples: There is inheritance and association present throughout the program. Such as the “Player” and “Crate” classes inheriting “MoveableTile” which in turn inherits “Tile”. |

|  |
| --- |
| **Parameters are passed correctly both within and between objects** |
| Examples: There is a large amount of parameter passing present, such as the parameters from “Gameplay” to “Level” class, or the methods in “MoveableTile” parameter passing to each other. |

|  |
| --- |
| **Appropriate access types are defined for methods, attributes and classes** |
| Examples: All attributes are made private with public facing getter and setter methods used to access them. But only for the attributes that need to be shown outside the class. |

|  |
| --- |
| **Use of pre-defined classes and/or methods from the standard object library** |
| Examples: JavaFX’s library was used in order to create the visual objects shown on screen and the methods to interact with those objects too. A lot of default java libraries were imported as well |

|  |
| --- |
| **The program appropriately handles errors with exceptions or pre-validation** |
| Examples: There is try-catchs in the program to catch any errors that can occur. Such as the try-catchs in “Gameplay” that is used to verify if the files in the level folder are correctly labelled. There is also some “throws” so some functions can be handled correctly in case of error. |

|  |
| --- |
| The program code is commented appropriately throughout |
| Examples: There is a large amount of comments describing what methods do, there is also comments to help with the readability such as marking the attributes, constructors and methods into their own sections |