# 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 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** |
| Used private member variables so that they can only be used within the class. Getters and Setters are then used so that they can be accessed via other classes.  Abstraction is not used thus far. There has not been any need to hide unwanted complex code.  If it were to be used, it would be used on a class that has multiple subclasses. This could mean that MapElement could potentially become abstract. |

|  |
| --- |
| **Inheritance used if appropriate to the solution** |
| My classes of Player, Wall, FloorSpace, crate and Diamond all inherit from MapElement.  Examples:  **public** **class** Wall **extends** MapElement  This means that the class Wall and the class MapElement is in a “Is-a” relationship.  Wall ***is-a*** MapElement  This practices good code reusability. |

|  |
| --- |
| **Polymorphism used if appropriate to the solution** |
| So far, no method overloading or overriding has been used.  However, when coding entity movement, classes like wall and crate inherit from the same class – MapElement – but crate will override movement. |

|  |
| --- |
| **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** |
| The arraylist of listOfTiles in the Level class is private, which means that it can only be accessed via the getter getListOfTiles elsewhere. |

|  |
| --- |
| **Data coupling is used (parameter passing) in preference to content or common coupling** |
| Examples: |

|  |
| --- |
| **Program does not contain a lot of unnecessary data coupling** |
| Examples: |

|  |
| --- |
| **Classes are highly cohesive** |
| Examples:  So far the classes are cohesive. SokobanMapReader will only read the map input, SokobanGame class then uses the returned value from SokobanMapReader to allow the user to select the desired level, then the level class uses this input to generate the selected level. |

## Implementation requirements:

|  |
| --- |
| **A working solution which meets the requirements of the given brief** |
| Comments:  No working solution as of yet. Very limited code. |

|  |
| --- |
| **Variables are correctly declared and initialised** |
| Examples: |

|  |
| --- |
| **Arithmetic and/or logical operators are used correctly** |
| Examples: |

|  |
| --- |
| **A range of control structures are implemented correctly** |
| Examples: |

|  |
| --- |
| **At least two data structures are implemented correctly** |
| Examples: |

|  |
| --- |
| **The program contains a minimum of four classes, which contain attributes, methods and a constructor method** |
| Examples: |

|  |
| --- |
| **A minimum of three objects are created from the classes, with appropriate initial attribute values set through the constructor methods** |
| Examples: |

|  |
| --- |
| **The program contains at least one overloaded method (this may be a constructor method)** |
| Examples: |

|  |
| --- |
| **Classes are linked appropriately through association, aggregation or inheritance relationships** |
| Examples: |

|  |
| --- |
| **Parameters are passed correctly both within and between objects** |
| Examples: |

|  |
| --- |
| **Appropriate access types are defined for methods, attributes and classes** |
| Examples: |

|  |
| --- |
| **Use of pre-defined classes and/or methods from the standard object library** |
| Examples: |

|  |
| --- |
| **The program appropriately handles errors with exceptions or pre-validation** |
| Examples: |

|  |
| --- |
| The program code is commented appropriately throughout |
| Examples: |