Zuul Project Critique Report

S.No	Flaw	Consequences	Solution
1	Several methods in Game class use separate code to print the location information. It is a code duplication and thus results in poor cohesion.	Bad for code maintainability as any change in print statements needs to be done repeatedly to other places. It will create difficulty for the maintainability Programmer as it may make change in one piece of code and forget to make same changes to other method. It can also cause inconsistency in code in future due to above mentioned reasons.	To create a separate method for printing and call it wherever printing of location is required. It improves cohesiveness, code reusability and code maintainability.
2	Fields in Room class declared public causing other classes to directly access it. It is against the OOP design principle of encapsulation and thus causes tight coupling between Room and other classes.	Increased coupling of Room class with other classes as classes have accessed the Room fields directly in their business logic. Making any change to Room class's fields requires a lot of changes in other classes as well as it may introduces bugs.	Use encapsulation principle and make fields private in Room class. Use getter/setters to access Room fields in other classes to reduce tight coupling between classes.
3	Variables to represent Exits in Room class are defined as objects; one variable per exit. Increases complexity and is not a good designing practice.	Adding any new exit directions in the Room class in future will need introducing new variables which is not a good programming design as well as require a substantial amount of changes to the code to handle any new actions/directions in future.	To use a HashMap implementation to hold the exit directions of the Room. It will allow saving any more directions in future without any need to change code in Room class. Room class will also get simplified.
4	Method to print Room's description is in the Game class causing coupling between Room and Game class. It is against responsibility-driven design principle as printing any information about Room should be Room class's responsibility.	Adding any modifications in Room class will affect the Game class as well.	Move the Room's printing information to Room's class. Call method from Room's class to get the information wherever required.

5	Game class is printing commands which should be the responsibility of the CommandWords class. It again violates the responsibility driven design principle.	Adding any new commands in CommandWords class will also need changes in the Game class which can easily be overlooked as the coupling is implicit between the two classes.	Implement responsibility driven design principle. CommandWords should be responsible to print all valid commands and Game class should call a method from CommandWords to get that information. It will remove implicit coupling between two classes. Thus improves code maintainability and increases cohesion and reduce implicit coupling.
6	Hardcoding in many classes to print the output on console.	It will require a lot of changes in different classes if in future any different user interfaces are adopted for the game.	To encapsulate all the information about user interface in one class so that if in future game company wants to switch to any other user interface from text based UI, it can easily make changes in one class only. All other classes should return string to Game class and printing on console should only be done from Game class.
7	Item fields and Character fields are added in Room class. It is a bad design in terms of class cohesiveness as Room class should not hold item and character details. Items and Characters are a separate logical entities and hence should have separate classes.	Any changes in specification of Item or Character class will require change in Room class. It does not currently support to hold multiple items or multiple characters in a room which can be a possible future extension and would require substantial changes in future.	Solution is to create a separate class for Items and Characters and add Room class to hold references to these objects. Also , its better to use collections for items and characters in the Room class to support any possible future extension for a Room to hold multiple items or characters in the Room.
8	Game class contains fields to save information related to the game player. It is against the principle of responsibility driven design as information regarding player should be stored into a separate class which should be logical representation of a player.	Current flaw doesn't support for the company's future goals to add multiple players.	To create a separate class to represent a player in the game and add a reference to a collection of Players objects in game class to allow support for company’s future goal to add multiple players.
9	Information regarding current room in the game class is against responsibility driven design principle.	If multiple players are added in future , each player should have its own current Room. If current Room variable stays in the game class, all multiple players in future will have only one current room.	Move current Room information to Player class to improve responsibility driven design principle.

10	Separate collections for Items and their relevant weights.	Increases complexity in the code and makes code maintainability difficult. Not a good design principle.	Using more appropriate collection for the purpose of storing related information in a key-value pair i.e. HashMap instead of separate array lists.
11	Literal string used to invoke commands and to refer exits/directions in the whole code.	It causes implicit decoupling between several classes, causes code duplication, causes difficulty to add new commands/directions in future.	Create Enumerated types for the Commands and Exits.
12	User Interface/Game logic closely tied to commands in English. Game class comparing literal strings in English to invoke different commands. It is again implicit coupling.	If in future game needs to be translated to any other language, programmer will have to find all places where commands are referred and change it.	Create self-contained Enumerated types for the Commands words to store the actual text for the commands. Enums should then be referred in the code in a language-independent way.
13	CommandWords and Commands class names do not hint their purpose.	Creates difficulty for a programmer to understand the purpose of the two classes.	Change name of the class to improve understanding readability.
14	Game class tightly coupled with the logic to invoke commands.	Every time a new command is added, a new case needs to be handled in Game class. It causes tight coupling between Game and new commands.	Implement Command Design pattern to decouple the class(Game) invoking the commands from the classes responsible to execute them. It will also new actions/commands to be added efficiently and with minimal (or no) changes to code in future.
15	Current code is not restricting to create multiple instances of the Game class.	Multiple instances of game can be created which should not be allowed. All clients should have only one and single instance of game available which is logically correct.	Implement singleton design pattern in Game class which restricts creation of multiple instances of a class and enforce all clients to use the single instance.
16	No java package created in the code which is not a good coding practice.	Creating packages can later on be useful to implement access restrictions to the classes and to resolve namespace conflicts. It can also be used to bundle classes together.	Create a package and move all classes within packages to logically group the related classes together. In future, it can help maintenance programmer to understand the code.
17	Code to take a new item by a player is not logically correct.	Player will not be able to take any item.	Correct the condition that if weight of the new item and player's current weight is less than the maximum allowed weight that a player is allowed to take; let the player take the new item.

18	Bug in parser class as it does not return the third word of the command entered by the user.	Commands requiring third word of the command won't execute successfully.	Fix the bug by returning the third word entered by the user in the Parser class.
19	Implicit constructor not found in some classes.	Can create errors when creating class instance through reflection.	It is a good programming practice to include constructors even if the body is empty. Add constructors wherever missing.
20	Improper code indentation.	It reduces the readability of the code which can create difficulty for programmers that may work on the code on future. Improper indentation can also sometimes introduce some bugs.	Use proper indentation as defined by Java programming guidelines to improve code readability and maintainability.
21	Give Command tells us that Items can be given to Characters which shows there should be a relationship between the two classes but at present there is no relation between the two classes.	Give command will not work as Characters don't hold any relationship with Item class.	HAS-A relationship should be created between Character and Item class as Characters can have item or items(in future). Creating a HAS-A relationship by adding a collection of Items in the Character class.
22	Type not specified when declaring Collections to hold items and weights in Game class.	It is not a good programming practice and can cause runtime errors.	Collections are generic classes. It is a good programming practice to explicitly state which Object a collection is going to hold. Specify type in angle brackets.
23	Drop item method removing element from Array list without using Iterator.	Causes run time error.	When a collection needs modification during iteration, it must be done using an Iterator.
24	Parser class should have only one instance throughout the application but current code doesn't enforce it.	Multiple instances of parser class can be created which is incorrect logically as there should be only one parser /compiler in any application.	Implement singleton design pattern for the parser class.
25	Game class is coupled with the logic to create players which also shows poor cohesion.	If in future company needs to introduce other type of players or want to change the way players are created , then there will a lot of change in Game class.	Implement Factory design pattern and encapsulate the logic to create players into a separate class to provide loose coupling and high cohesion. It will also support company's goal to create Computer Controlled Players in future with minimum change in game class.

26	Player logic tightly coupled with the specific type of a player.	A lot of code handling and changes to introduce any different type of player in future which is not good for company's goal to introduce new Computer Controlled Characters/Players in future.	First create an abstract class that consists of all functionality that any future Player classes must have. It also ensures a structure that any future Player must have. Secondly, use the Polymorphic property of the Variables which enables them to hold objects of their declared type or any subtype of their declared type. Using variable of abstract class type of a Player will need no changes when any new type of players will be introduced in future as every new type of player will be a sub class of the abstract Player class. It also supports company's idea to introduce Computer Controlled characters in future with no change required in logic.
27	Exceptions not handled in the code.	It can create an unpleasant experience for the user during game play.	It is a good programming practice to handle exceptions and can also be used to show customized error messages to users in a more user friendly way.
28	Hard coded String literals are used within all classes to return any message to users.	If game needs to be translated in any other language other than English, a lot of work needs to be done to find all hardcoded string literals within different classes.	To keep all Strings in a separate file and access it from that file within class. If Internationalization needs to be done, string file will be given to a library to translate it.