Good Programming Practice

In relation to what I have learnt from this unit, this document will list what I consider as possible good practices in Object-oriented Programming.

1. **Utilize the principles of Abstraction, Inheritance, and Polymorphism to maintain a solid system and hierarchy among all classes:**
   When writing the first class (or classes in some cases) in a program, it is best to make them abstract classes, then use Inheritance to create derived classes and determine a class hierarchy, and after that, turn them into more specialized classes afterwards using Polymorphism.
   Doing this will allow you to expand the program later on much more easily, while at the same time maintain unity between all the classes within the program, making accessing objects of all classes more simpler as well.
   This has been demonstrated in the SwinAdventure program codes, where every other class simply derives from a single IdentifiableObject class, with all objects possessing unique identifiers. From the original IdentifiableObject class, the inheritance branches into either classes for "objects" within the game, the children of the GameObject classes, or classes for "commands" used to control the player character, the children of the "Command" class. Objects of all these classes are able to access each other using identifiers given to them, as they are, at the very core, all Identifiable Objects.
2. **Use Encapsulation to maintain integrity of data within objects:**
   If possible, the access level of all data fields within an object should be set to private so that they cannot be accessed directly from objects of unrelated classes, and to allow access method to them, either public or protected method should be created (depending on whether the classes you intend to grant access to these are child classes or not).
   This practice has been demonstrated throughout most of the programs in the semester. One good example would be the Planet Rover program. A Battery object contains a fCharge field that indicates the number of charges remains in it, and there is no way to change that value freely. A Device object can only either see the remaining charges, or drain them from the Battery.
   In some cases, access to how one can control an object can even be limited further by having methods that are not meant to be called from the outside set to private, especially methods that are used to modify data within an object.