Colin Murphy
Oscar Delgado
Object Oriented Analysis & Design Project 1
1/31/20

1.
For each one: Provide definition, how they apply to OO classes, examples of how they're good or bad

a. Abstraction - set of concepts that some entity provides you in order for you to achieve a task or solve a problem. In OO is by displaying essential information while hiding the background detail or the implementations. It good by reducing the programs complexity. (https://www.geeksforgeeks.org/abstraction-in-c/ and slides)
b. Encapsulation - The action of enclosing something in, this relates to OO by hiding implementation details of a task/algorithm through layering of accessibility in classes and subclasses(Quiz). Encapsulation is good because it stops certain classes from accessing variables they're not supposed to. Encapsulation can be bad because if done poorly it can lead to unwanted dependencies in your code.
c. Polymorphism - Literally means "many forms" this applies in OO as an object can refer to a method different derivations of a class in the same way but get the behavior appropriate to the class being referred
d. Cohesion - refers to how closely the operations in a routine are related (Slides). Good Cohesion in code is if it exhibits strong/highly cohesion meaning methods perform one operation, classes achieves a fine-grain goal. Week cohesion is bad because if a method does more than one operation than if you ever have to go back and change the code you would have to change so much more.
e. Coupling - The pairing of items, this relates to OO because it refers to what your classes are connected to and responsible for(Slides). Tight coupling is bad because it means that changes in your code have an unwanted rippling effect, potentially causing bugs. Loose coupling is good because when you change your code you don't have to worry about everything being affected, your code responds well to change
f. Identity - Being who you are, it is related to OO because all objects have a unique id(Slides). Identity is good because it makes all of your objects unique and callable if you desire to use them. It also lets you know if two variables point at the same object. Identity can be bad because it can be hard to come up with so many clever names

2.
Functional Decomposition approach is when you take a big problem and decompose it into smaller problems until you can solve it in easy steps.
When approaching this problem I am assuming there is a database for the employees and in the database has the hours worked for the month and their names and addresses and hourly pay..
Payroll systems
|

Database search
|
Employee name ----- Hourly pay ------hours worked ------address
| |
Hourly pay * hours worked
|
Check for employee name of amount ^ sent to address

3.Object-Oriented approach.
-For our payroll there will first be a class ‘Employee’ which will contain things that all workers have such as Employee ID, Name, Address, and Hours Worked (Assume they’re all paid hourly)
-Then, because there are different jobs in our company, each type of job will be a subclass of ‘Employee’. Ex. ‘Secretary’, ‘Salespeople’. These subclasses will contain the unique hourly pay for the position
-Then each employee is an object of their job position’s class, which again is a subclass of ‘Employee’.
-Have a ‘Paycheck’ class with association to the ‘Employee’ class that actually constructs the paychecks. Using the personal information from ‘Employee’, the hourly pay rate from the ‘Employee’ subclasses, and with the information coming from the employee objects. The finished paychecks can be sent out.