**Object-Oriented Exercise Overview and Outcomes (1220: TOOP)**

**United States Coast Guard Academy**

**15 August 2021**

**Object-Oriented Exercise Proposal**

In order to better enable students to digest the complex fundamentals behind introductory object-oriented programming (OOP), I propose that students complete a group exercise in which they build object-oriented models for common relationships.

The groups would be comprised of two-three (maximum four) students each in order to ensure that each student has a role to play. In some cases, partners may be preferable to encourage active participation. The exercise would take approximately 20 to 30 minutes to complete, following which students would be encouraged to demonstrate their models to the class as a whole. For this reason, each group would receive a unique relationship to model.

Groups should be guided towards ‘discovering’ concepts such as encapsulation, abstraction, and inheritance either directly via problem design or via active guidance from instructors and teacher’s assistants.

**Outcomes**

By enabling and encouraging students to develop their own object-oriented models for relationships they might encounter in day-to-day life, students might be led to develop a certain intuition for object-oriented design. In the best case, this will enable students to better appreciate the principles that guide modern OOP as they learn more and more later on. In the minimum case, students will be made familiar with the key object-oriented concepts encapsulation, abstraction, and inheritance.

Polymorphism likely falls outside the scope of which students might discover themselves, and it may be beneficial for an instructor to highlight where polymorphism might fit in models following the exercise.

**Problem Structure**

Each group should be presented with a relationship they are familiar with from day-to-day life. An example might be one of the following:

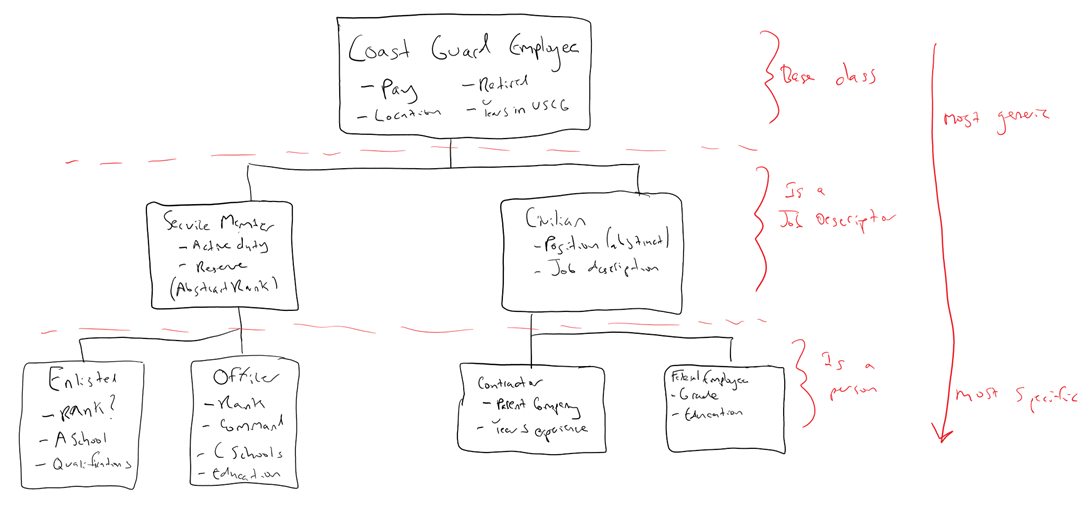
* Continents à Countries à States à Cities
* Boats à Civilian, Military à High Endurance, Medium Endurance, Small
* Coast Guard Employees à Military and Civilians à Enlisted and Officer, Contractor and Federal Employee…
* Board Game Types à Strategy, Tower, Financial, Military, Role Playing à …
* Computer Game Types à Strategy, First Person, Muli-player Cooperative, Tower, Military, Role Playing, Card à …
* Animal à Domestic, Wild à Pet, Farm, …
* Bodies of Water à Oceans, Lakes, Rivers à …
* Land Vehicle à Car, Pickup, Truck à Sedan, SUV, Crossover, Toy Pickup, Full-size Pickup …
* Academic Course à Fields (Science, Math, Arts, etc.), HAS-A Grading System…
* Engineer à Civil, Mechanical, Naval Architecture, Electrical, Network, HAS-A Qualification, …
* Bank
* Stock Market

Students should be presented with a guiding example at the beginning of the lecture. The example should be made readily available (physical per group handout or posted on a projector for the duration of the exercise) in order for students to reference as they develop their model. Unified Modeling Language (UML) diagrams would be an ideal way of conveying this example.

Diagram

Description automatically generated

**Figure 1.** UML Descriptor of a Coast Guard Employee Relationship[[1]](#footnote-1)



**Figure 2.** Handwritten Relationship of Coast Guard Employees

**Considerations**

It is important that students make the connection between the diagrams they are writing and the concepts they will eventually be using in day-to-day programming. With that in mind, it may be prudent for students to be introduced to OOP terminology before they begin the exercise, or at the very least at the end. The focus of the exercise is to build intuition; giving a name to the intuitive concepts they might’ve embodied following the exercise could prove beneficial later on.

It may be wise to save student’s diagrams following completion of the exercise in order to refer back to later on in the semester. Furthermore, it may be prudent to consider the example given in the exercise as a focal point for lectures in the future – as a reference to show how the lesson applies to an already-familiar relationship.

**Appendix A**

|  |
| --- |
| @startuml  !theme crt-amber  @startuml  class "Coast Guard Employee" {  double pay  bool retired  Location station  int yearsServed  string name  int age    ReceivePaycheck()  Retire()  Move()  }  class "Service Member" {  bool activeDuty  Date nextPromotion    displayUniform()  promote()  }  class "Enlisted" {  EnlistedRank rank  List qualifications  }  class "Officer" {  OfficerRank rank  bool command  List<CSchool> schools  Education education  }  class "Civilian" {  string jobDescription    promote()  }  class "Contractor" {  Company parentCompany  int yearsExperience  Field expertise  }  class "Federal Employee" {  Grade employeeGrade  Education education  Role role  }  class "Rank <Abstract>" {  string Descriptor  }  class "Enlisted Rank" {  int grade  ASchool rate  }  class "Officer Rank" {  int rank  bool flag  }  "Coast Guard Employee" --> "Service Member"  "Coast Guard Employee" --> "Civilian"  "Service Member" --> "Enlisted"  "Service Member" --> "Officer"  "Civilian" --> "Contractor"  "Civilian" --> "Federal Employee"  "Rank <Abstract>" --> "Enlisted Rank"  "Rank <Abstract>" --> "Officer Rank"  "Officer" --\* "Officer Rank"  "Enlisted" --\* "Enlisted Rank"  @enduml |

1. Made in <https://plantuml.com/> [↑](#footnote-ref-1)