# Section 4: Responsibility Driven Design

## Question 1

Describe what is meant by a ‘conceptual class’?

Does an Entity or Controller make a better example of a conceptual class? Explain why.

### Answer

A ‘conceptual class’ is a class that is not specifically mentioned in the requirements or use cases, but which are invented by the designer to fulfill a responsibility implied by the requirements or use cases.

Controllers are good examples of conceptual classes as they are invented to implement the application workflows implied by use cases. Delegate classes are another good example of conceptual classes. The delegates are invented to implement some service which is needed to fulfil a feature in the requirements (usually for a controller). For example, the OrderCheckout workflow (controller) needs to calculate the taxes owed for the customer’s purchases. TaxCalculator would be a delegate class which implements the tax calculations needed by the OrderCheckout Controller to complete the order processing.

## Question 2

According to Responsibility Driven Design, what two questions should the designer ask when evaluating the trying to identify a class to implement some feature or service identified in the requirements?

### Answer

What class is responsible for providing (doing) the service?

What class is responsible for knowing some information the system manipulates?

## Question 3

Describe the four goals of the design workflow.

### Answer

1. Extend the analysis models to implement the system features and responsibilities as services in design models.
2. Decompose the broadly defined analysis classes into cohesive, loosely coupled design classes ready for implementation.
3. Identify opportunities to apply architectural and design patterns, and other means of standardizing and improving the design.
4. Estimate time and manpower resources needed to execute the design and establish a firm delivery date.

## Question 4

What should a developer do when they identify a responsibility that does not *seem to* belong to any of the classes currently found in their design?

### Answer

When a responsibility is identified that can’t be logically assigned to an existing class it signifies that a new class should be created and assigned the responsibility.

## Question 5

Describe the three Responsibility Driven Design class roles found in most software designs? Hint: Identified as a Design Pattern in the slides.

### Answer

Boundary Classes are used to create a “presentation” of the model classes e.g. a screen with a form or report that the user interacts with as the user interacts with the system to obtain their goals. Also called Presentation or View classes.

Entity Classes are ‘things’ in the problem domain. These classes are CRUD by controllers and in most cases are persisted. Entities are easy to identify and are extracted from the requirements or use cases e.g. Customer, Product, etc. Also called Domain or Model classes.

Controller Classes implement / manage a workflow in the design from start to finish. E.g. a class that implements a use case scenario. Each workflow has a number of steps and decisions that are made as the workflow is executed. These steps and rules are encoded in the controller class design. Typically, there is a 1-1 between workflows and individual controller classes.

## Question 6

Name and describe the two types of responsibilities suggested by Responsibility-Driven Design and in the slides?

Which type of responsibility is assigned to Entity Classes?

Which type of responsibility is assigned to Controller Classes?

### Answer

Knowing and Doing Responsibilities

Knowing Responsibilities: A knowing responsibility describes a situation where the system is responsible for organizing and/or maintaining information. This usually implies, but not always, that the system needs to persistently maintain the information. Knowing responsibilities are assigned to Entity classes.

Doing Responsibilities: A doing responsibility describes a situation where the system is responsible for taking some action or producing some result. Doing responsibilities are usually assigned to Controller or Service classes.

## Question 7

What are five types of Controller Responsibilities commonly assumed by a controller class?

### Answer

1. Creating new instances or updating entity objects.
2. Creating new instances or updating view (boundary) objects.
3. Coordinating complex interactions between collaborating entities i.e. user and application.
4. Validation of information exchanged between the user and the application.
5. Persisting new and updating entity objects.

## Question 8

What is the difference between intelligent or dumb controller classes?

Why do we want our controller classes to be dumb?

How do we keep our controller classes dumb?

### Answer

A controller’s intelligence is an indicator of the amount of responsibility the class implements. An intelligent class directly implements its responsibilities in its code / logic. A dumb controller implements few of its responsibilities. But this does not mean that dumb controllers do not assume their responsibilities. Rather a dumb controller delegates the implementation of those responsibilities to its delegate service classes.

An intelligent controller is a controller that implements many of its responsibilities. This indicates a large and complex class that lacks cohesion and so is difficult to understand, modify, reuse, etc.

*Controllers should know what to do (and in what order) but not how to do*. Dumb controllers should implement the logic of a workflow, but delegate the implementation of workflow steps to delegate (service) classes.

## Question 9

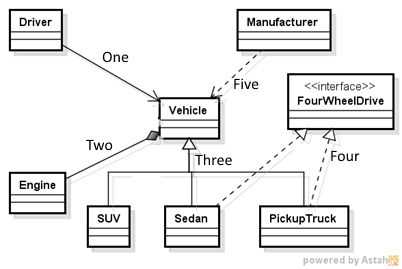
Does Concentrated Intelligence promote or undermine code reuse? Explain your answer.

### Answer

Concentrated Intelligence implies that the class (usually a controller) is implementing most of the services it needs to fulfil its responsibilities. When these services are required in multiple classes, implementing the same service in each class diminishes code reuse. Instead dumb controllers delegate to service classes and so create services that can be used by multiple controllers.

## Question 10

Identify each of the five associations in the following UML class diagram.



### Answer

1. Association (uses)
2. Composition / Part of
3. Extends / Inheritance / Kind of
4. Implements (Interface) Not the use of a stereotype to denote the interface.
5. Dependency (depends-upon or has-knowledge-of). A weak form of Association.