Key Object Oriented Concepts

Name: Minh An Nguyen

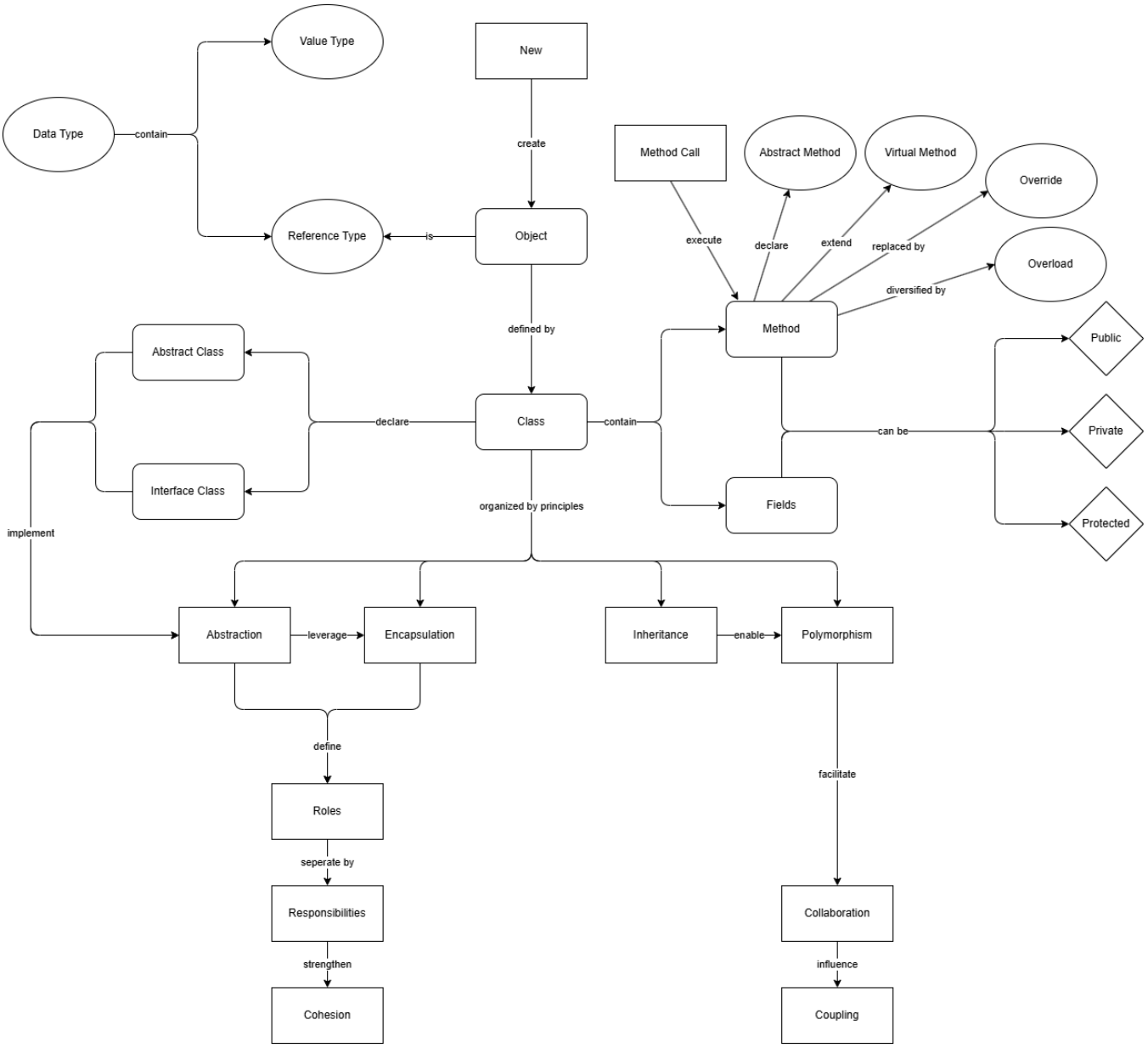
Student ID: 104844794

Object-Oriented Programming (OOP) is a way of writing programs by using objects. Objects represent things in the real world and have two main parts: properties (what they are like) and methods (what they can do). OOP helps make programs easier to manage, reuse, and update, especially when the project is big. There are four main ideas in OOP: Encapsulation, Abstraction, Inheritance, and Polymorphism. I will explain these principles and use the Drawing Program that I have developed as example:

1. **Encapsulation:** It is about putting data (like fields) and actions (like methods) into a single package, called a class. This keeps the data safe from unwanted changes. In the **Drawing Program**, each shape (like a **rectangle** or **circle**) has fields like color and position. These details are hidden inside the shape, and can only be changed or accessed by specific public properties like **Color** or **X** and **Y**. This makes sure that only the right parts of the program can change the shape's details.
2. **Abstraction:** It is about simplifying complex ideas by focusing on important parts and hiding the unnecessary details. In the Drawing Program, the **Shape** class is an abstract base class. It sets up general methods like **Draw()** and **IsAt()** that all shapes need, but it doesn't explain how they work. This lets us focus on what shapes should do without worrying about the exact steps for each shape. Other classes like **MyRectangle** and **MyCircle** then explain how these methods should work for their specific shapes.
3. **Inheritance:** It allows one class to get properties and methods from another class. This helps reuse code and manage related classes easily. In the Drawing Program, **MyRectangle** and **MyCircle** inherit from the **Shape** class. This means they get properties like **Color**, **X**, and **Y**, and methods like **DrawOutline()** from **Shape**. This makes the program more efficient because we don't have to rewrite the same code for each shape. We can also use objects of **MyRectangle** or **MyCircle** wherever a **Shape** is needed.
4. **Polymorphism:** Polymorphism means that methods can work in different ways depending on which object uses them. In the Drawing Program, the **Draw()** method is polymorphic. Even though the program might just see a shape as a general **Shape**, it will know to use the specific **Draw()** method for a rectangle or a circle when needed. This lets the program handle all shapes in a similar way, while still allowing each shape to act differently when drawn.

In the Drawing Program, these OOP ideas work together to create a program where shapes can be drawn in a flexible and organized way. **Inheritance** creates a structure where different shapes (like **MyRectangle** and **MyCircle**) are related. **Polymorphism** allows the program to handle any shape in a consistent way, even if they behave differently. **Encapsulation** keeps each shape's details hidden and safe, while **Abstraction** focuses on the important parts of how shapes should work, without getting into unnecessary details.

OOP Concept Map



Three examples of when we should use exception handling:

1. When the program tries to open a file, it might not always work. For example, the file might not exist or the computer might not let the user access it. In this case, we can use exception handling to make sure the program doesn't crash. Instead, it can show an error message.
2. If the program asks the user to enter a number, they might enter something else, like text. This can cause an error. Using exception handling, we can catch this mistake and tell the user to enter a valid number.
3. When the program does division, dividing by zero will cause an error. We can handle this by using an exception and prevent the program from crashing.