Research:

*What are the four pillars of Object-Oriented Programming? Explain each pillar.*

1. Abstraction

* Exposing essential features of an object and hiding those that don't matter that much

2. Encapsulation

* Hiding inner working of the code so it cannot be manipulated by the users or so that public features are not exposed to the users and used in a manner not intended for
  + example accessing object properties using dot notation is very bad practice
  + instead use access modifiers
    - Public - accessible everywhere
    - Private - only accessible within the class
    - Protected - accessible within the class, other classes in the same package, and all subclasses
    - No modifier - same as protected except is it is not accessible by subclasses in a different package
  + we encapsulate our properties so they do not change in a way we don't want them to
  + we access data using getters and setters

3. Inheritance

* Allows objects of classes to inherit from parent classes

4. Polymorphism

* One method of a class can take different forms for different objects
* animals communicate in different ways but they all communicate (method)

*What is the relationship between a Class and an Object?*

A class specifies the blueprint for how objects will be built including their properties and the functionality that each object will have. An object is an instance of a class.

*What are the differences between abstract classes and interfaces? When should you use one over the other?*

Classes can only inherit from one abstract class but they can implement multiple interfaces. Abstract classes have method definitions (method declaration and method body) while Interfaces only allows you to declare the methods with no body. All the methods declared in interfaces have to be implemented by the classes that implement that interface. All the methods in an abstract class are accessible to the subclasses that inherit from it. Abstract classes are used to define properties and methods that describe an object and what it is. Interfaces, however, are used to establish a contract between an interface and multiple classes that defines what those classes can do. Also use interfaces when we want to solve the issue of multiple inheritances or multiple classes implementing the methods in the class/interface.

*Research the SOLID principles of Object-Oriented Programming (OOP) as introduced by Robert Martin. List the principles, and give a description of each one.*

These are rules aimed to improve maintainability and design of software.

Single responsibility principle:

A class should only have one responsibility and should contain all the methods and properties to accomplish that responsibility.

Open closed principle:

Classes, functions, modules and other software entities should be open for extension but closed for modification. Through inheritance we can extend parent classes through creating subclasses which leave the parent class intact. Through polymorphism we can have the same method which is implemented differently throughout different subclasses.

Liskov substitution principle:

Subclasses should be able to maintain behavior of their parent entity as well as their new functionalities and properties.

Interface segregation principle:

Classes should only have functionality that it can use to achieve its end purpose/responsibility.

Dependency inversion principle:

High-level modules should not depend on low-level modules. Instead, both should depend on abstractions (interfaces). Abstractions should not depend on details. Details (like concrete implementations) should depend on abstractions.

*What is an exception? What are the differences between checked and unchecked exceptions?*

Exceptions events that disrupt the normal flow of execution of code. They cause the code to crash.

Checked exception - Events we want to explicitly check in our code. We can handle them if and when they occur. Events that we don't always have control over

Unchecked exception - an exception that we don't check for. Occurs because of coders mistakes. They can be avoided by writing clean code and we always have control over these types of exceptions.

*What is unit testing and why is it important?*

Tests that are written to test the fitness of new code in production. It is important for catching bugs that may break production environment.