Q1.c

Polymorphism

Polymorphism refers to the design that enables an object or a method to take on multiple forms. It is also applicable to methods where they can take on multiple forms depending on some factors.

In the context of class, Polymorphism allows us to use an instance of the child class to initialize or assign to a parent class reference variable, This type of polymorphism is also known as subtype polymorphism, as the child class will be a subtype of the parent class.

The two other types of polymorphism are - Runtime polymorphism and Compile-time polymorphism. Runtime polymorphism occurs when a method in the parent class is overridden by the child class. Now, the function call will be resolved at runtime depending on the instance being used. It is also known as Function overriding. Compile time polymorphism is function overloading. It occurs when the same function name is used in the multiple function definitions. Which function to be called will be resolved depending on the number and type of parameters used in the function call.

Good use:

One of the good uses of polymorphism is that it can refer to interfaces instead of implementations. Here the interface acts as a parent class in terms of its role, but it is not a class. In the below code, if we use a specific implementation of List<String> such as ArrayList or LinkedList as the return type, then the method getAllModelNames() is restricted to use that specific implementation. Instead, if we use List<String> then getAllModelNames() is given the choice of using any class implementation according to its requirement.

```
class HondaCar {
    public List<String> getAllModelNames() {
        // method definition
    }
}
```

Bad use:

Suppose the requirement is that we want to restrict the child class from overriding some methods of parent class. We cannot achieve this because of the runtime polymorphism capability available in Object-oriented programming (although we could use the “final” keyword to satisfy our requirement here). So, polymorphism is not helpful here.

```
class A {
    public void method1() {
        // Method definition
    }
}
```

```
class B extends A {
    public void method1() {
        // Method definition
    }
    // Here method1() is being overridden by the child class B
}
```

(Web Reference used for this answer - https://www.tutorialspoint.com/java/java_polymorphism.htm)

Q1.d

Cohesion

Cohesion indicates the degree to which a class or function has a single, well-focused purpose. In the context of Object-oriented classes, if a class has low cohesion, then it means that it is performing multiple tasks. On the other hand, if it has high cohesion, then it means that it has one single and well-focused responsibility. Higher cohesiveness leads to better design. The same is applicable to methods or functions in the program.

Bad use of cohesion:

The below code snippet shows a class with less cohesiveness which has the responsibility of multiple unrelated tasks. Here, HondaCrv class maintains a list of the customers who bought it. Logically, the task of maintaining customer names should not be the responsibility of HondaCrv class. So this class has low cohesion.

```
class HondaCrv {
    private List<String> customerNames;
    public HondaCrv() {
        customerNames = new ArrayList<>();
    }
    public List<String> getCustomers() {
        return customerNames;
    }
    public void addCustomer(String customerName) {
        customerNames.add(customerName);
    }
    // other variables and methods definition of HondaCrv class
}
```

Good use of cohesion:

One of the good use of cohesion is where the class has high cohesion and has only one and well-focused responsibility. The ‘HondaCrv’ class has the behavior and properties related to Honda CRV car only. The

HondaCrvCustomer class has the task of maintaining a list of customers who bought Honda CRV. The Customer class is a more high-level class that maintains the details of a single customer. All these classes have only one clear responsibility and hence have high cohesion.

```
class HondaCrv {
    public HondaCrv() {
        // initializations done here
    }
    // other variables and methods definition of HondaCrv class
}

class Customer {
    private String name;
    public Customer() {
        // initializations done here
    }
    public void setName(String name) {  this.name = name;  }
}

class HondaCrvCustomers {
    private List<Customer> customers;
    public HondaCrvCustomers() {
        // initializations done here
    }
    public List<Customer> getCustomers() {
        return customers;
    }
    public void addCustomer(Customer customer) {
        customers.add(customer);
    }
}
```