Name : Imran Hamid
ID : IT-22010

01. When to use interface and when to use abstract class. Develop a story.

⇒ Suppose you're developing a system for a restaurant. This restaurant has different types of employees. For example, there's a chef who prepares delicious food, a Cashier who handles customers payments, and a Delivery Boy who delivers foods order.

Some of the employees has extra talents. For instance, some can sing to entertain customers and some can drive vehicles to deliver food. However all employees share certain responsibilities - they must clock in for work and perform their main job duties.

In this context, when designing software we decide where to use interface and where to use abstract classes. For common properties and

behaviours shared by all employees, such as having a name or the ClockIn() method, we use an abstract class. This help us avoid code repetition and keeps shared logic in one place.

On the other hand, for optimal skills or behaviours that not employees have - like singing or Driving we use interfaces. Theseses represents abilities that only specific employees might implement, making them perfect for interfaces.

IT-22010

Code:

```
public interface singer {        // interface optimal skills.
    void sing();
}
public interface Driver {
    void drive();
}
public abstract class Employee {
    protected String name;
    public Employee (String name) {
        this.name = name;
    }
    public void ClockIn() {
        System.out.println(name + "Clocked in");
    }
    public abstract void doJob();
}
```

// Chef can.. sing

IT-22010

```
public class Chef extends Employee implements Singer {
    public Chef (String name) {
        super (name);
    }
    @override
    public void doJob () {
        System.out.println (name,+ " is cooking Chicken");
    }
    @override
    public void sing () {
        System.out.println (" name + " sings Chicken tandoori song");
    }
}

public class Cashier extends Employee {        // cashier, special no skills
    public Cashier (String name) {
        super (name);
    }
    @ override
    public void doJob () {
        System.out.println (name + " is taking orders");
    }
}
```

(T-2201.

```
//Delivery boy can drive & sing
public class DeliveryBoy extends Employee implements Driver, Singer
{
 public ~~void~~ DeliveryBoy (String name){
    super (name);
 }
@ override
 public void doJob () {
   system.out. println (name + "is delivering chicken");
 }
@override
 public void drive (){
  system. out. println ("name + " is driving cycle ");
 }
// @ ~~public~~ override
 public void sing (){
 system.out. println ("name + " sing something");
 }
}
```

17.2.2010

```
public class Restaurant {
    public static void main (String[] a) {
        Employee chef = new Chef ("Imran the chef");
        chef.clockIn();
        chef.doJob();
        ((Singer) chef).sing();
        System.out.println();

        Employee cashier = new Cashier ("Logan the cashier");
        cashier.clockIn();
        cashier.doJob(); // no sing or drive here
        system.out.println();

        Employee delivery = new DeliveryBoy ("Rose the rider");
        delivery.clockIn();
        delivery.doJob();
        ((Driver) delivery).drive();
        ((Singer) delivery).sing();
    }
}
```