Name: Abdullah Al Mamun Sagor
ID: IT22059

1. When to use interface and when to use Abstract Class ?

Story: I am building software for a zoo. Different animals have different behavior. Some animals can fly, some can swim an some are mammds.

- All animal should have a makeSound() behavior.
- Animals like Birds and Fish behave differently they don't share a common superclass behavior for flying/swimming
- But all mammals share some base behavior like breathing through lungs.

Key point:

1. Use abstract Class Animal to define common behavior or states (breathing/sleeping).
2. Use interface like Flyable, Swimmable to add capabilities that cut across class hierachies.

IT22059

Java Code:

```
Interface Flyable {
    void fly();
}
interface Swimmable {
    void swim();
}
abstract class Animal {
    String name;
Animal (String name) {
    this.name = name;
}
  void sleep() {
    System.out.println(name + " is sleeping...");
}
    abstract void makeSound();
Class Bird extends Animal implement Flyable {
    Bird (String name) {
        Super (name);
    }
    @ override
    public void fly() {
System.out.println(name + " is flying in the sky!");
}
```

```
@Override
  void makeSound(){
   System.out.println(name + " Chirps!");
Class Fish extends Animal implement Swimmable{
    Fish(String name){
      super(name);
    }
  @Override
   public void swim(){
     System.out.println(name + " is swimming in water!");
    }
  @Override
   Void makeSound(){
   System.out.println(name+" makes blub-blub!");
  }}
}

Class Dog extends Animal{
    Dog(String name){
    }super(name);
  @Override
    void makeSound(){
   System.out.println(name + " barks!");
  }
}
```

IT22059

```
Public class Zoo {
    public static void main(String[] args) {
        Bird parrot = new Bird("Parrot");
        Fish goldfish = new Fish("Goldfish");
        Dog dog = new Dog("Buddy");
        parrot.sleep();
        parrot.fly();
        parrot.makeSound();
        goldfish.swim();
        goldfish.makeSound();
        dog.sleep();
        dog.makeSound();
    }
}
```

2. Are interface method slower than Abstract class method?

Ans: Not significantly in modern JVM's. Early version of Java had slighty slow method resolution for interface due to dynamic dispatch But modern JVM use JIT compilation and method lining, makes the difference negligible.

Java Code:

```
Interface Engine {
  void start();
}
abstract class vehicle{
  abstract void start();
}
Class InterfaceCar implements Engine{
  public void start(){
System.out.println("Starting car with abstract class...");
}}

public class PerformanceTest {
  public static void main(String[] args){
    long startTime, endTime;
Engine car1 = new InterfaceCar();
Vehicle car2 = new AbstractCar();
 startTime = System.nanoTime();
  for int i=0; i<1 000 000; i++){
  }  car1.start();
    endTime = System.nanotime();
    System.out.println(" Interface method time;
                    "(endTime-startTime)+" ns");
```

```
startTime = System.nanoTime();
    for (int i=0; i< 1000000 ; i++){
      car2.stat();
    }
    endtime = System.nanoTime();
    system.out.println(" Abstract class method time:
                " + (endtime - startTime)+ "ns");
  }
}
```

Note: This is synthetic benchmark not reliable for real word conclusions. In practice the difference is trivial dute to JVM optimizations.

IT22059

3. Abstract Class vs Interface – Comparison Table

Feature	Abstract Class	Interface
Purpose	Code reuse and common behavior	Define contract or capablilities
Speed	Abstract classes are fast	Interface are slow
Constructor	yes	No
Multiple Inheritance	Not allowed	Allowed (a class can implement multiple
Default method body	Yes (can have method body)	Yes (from Java 8 using default
Access Modifiers	Can use any (public protected etc)	All methods are public by default
Fields / variable	Can declare instance variable	Only static final constants
Example	abstract class Animal	interface Flyable