IT22049

Ans-1: At the Magical Creatures Academy, there are different creatures. All creatures must have a way to move so a common is required. Some animals also have special ability like being able to glow in the dark.

Use abstract class: To define common attributes and basic behavior shared by all animals (like name, move)

Use interface: To define additional abilities that may apply to some animals (like glowing)

Code:

```
interface Glowing {
        void glow(); }

abstract class Animal {
        string name;
        public Animal (String name) {
            this.name = name; }
        abstract void move();
        public void showName() {
            System.out.println ("Animal name : " + name); }

}
```

```
class Firefly extends Animal implement Glowing {
    public Firefly (String name) {
        super (name); }
    public void move (){
        System.out.println (name + " flies through the night
    }
    public void glow (){
        System.out.println (name + " glow with a soft yellow
                                                    light");
}
class Elephant extends Animal {
    public Elephant (String name) {
        super (name); }
    public void move (){
        System.out.println (name + " walks slowly with heavy
                                                    steps");
}
public class MagicalCreatureAcademy {
    public static void main (String [] args) {
        Animal a1 = new Firefly ("Twinkle");
        a1.showname();
        a1.move();
        ((Glowing)a1).glow();
        System.out.println();
        Animal a2 = new Elephant ("Dumboo");
        a2.show(Name);
        a2.move();
    }
```

Ans 2: In very old versions of Java, interface method calls were slightly slower because of how JVM handled them. But since Java 7++, JVM improvements so many thing that it make the difference practically zero. Only if calling a method billions of times in a loop make interface slightly slower. In 99.9% of real-world code performance difference between inteface and abstract class method calls are negligible.

Example:

```
interface MyInterface {
    void show();}
abstract class MyAbstract {
    abstract void show();}
class InterfaceImpl implements MyInterface {
    public void show (){}
}
class AbstractImpl extends MyAbstract {
    public void show();{}
}
Public class Interface_Vs AbstractSpeed {
    public static void main (String[] args) {
        myInterface obj1 = new InterfaceImpl();
        myAbstract obj2 = new AbstractImpl();
```

```
long start, end;
start = System.nanoTime();
for (int i=0; i< 1000-000; i++){
        obj1.show(); }
end = System.nanoTime();
System.out.println("Interface method time: " + (end-start)+" ns");
start = System.nanoTime();
for (int i=0; i<1-000-000; i++){
        obj2.show(); }
end= System.nanoTime(); }
}
```

Ans3:

Feature	Abstract Class	Interface
Method type	Abstract & concrete metho	Abstract (default, static) form Jam
Constructor support	Yes	no
Object creation	Can't create directly	can't create directly
Multiple Inheritance	Not supported	Supported
Inheritance keyword	extends	implements