```
//Design Pattern
Assignment

//1

final class Singalton{
     private static final Singalton INSTANCE = new Singalton();
     private Singalton() {
          if (INSTANCE != null) {
               throw new IllegalStateException("Already instantiated");
          }
     }
     public static Singalton getInstance() {
          return INSTANCE;
     }
     public Object clone() throws CloneNotSupportedException{
          throw new CloneNotSupportedException("Cannot clone instance of this class");
     }
}
class SingaltonDesign{
     public static void main(String[] arg){
          System.out.println(Singalton.getInstance());
          System.out.println(Singalton.getInstance());
     }
}


2.
abstract class Polygon {
     public abstract int getPolygon();
     public String toString() {
          return "Polygon= " + getPolygon();
     }
}
class Triangle extends Polygon{
     private int sides;

     Triangle(int sides){
          this.sides=sides;
     }
     public int getPolygon(){
          return this.sides;
     }
}
class Rectangle extends Polygon{
     private int sides;
     Rectangle(int sides){
          this.sides=sides;
     }
     public int getPolygon(){
          return this.sides;
     }
}
class PolygonFactory {
     public static Polygon getComputer(String type,int sides){
          if("Triangle".equalsIgnoreCase(type)) return new Triangle(sides);
          else if("Rectangle".equalsIgnoreCase(type)) return new Rectangle(sides);
          return null;
     }
}
class Polygon2{
     public static void main(String[] args) {
          Polygon triangle = PolygonFactory.getComputer("Triangle",3);
          Polygon rectangle= PolygonFactory.getComputer("rectangle",4);
          System.out.println("Factory Triangle sides::"+triangle);
```

```
        System.out.println("Factory Rectangle sides::"+rectangle);
    }
}


3.
enum CarType {
    MICRO, MINI, LUXURY;
}
enum Location {
    AUS, USA, INDIA;
}
abstract class Car {
    CarType carType;
    Location location;
    public Car(CarType carType, Location location) {
        this.carType = carType;
        this.location = location;
    }
    abstract void construct();
    @Override
    public String toString() {
        return "Car{" +
                "carType=" + carType +
                ", location=" + location +
                '}';
    }
}
class LuxuryCar extends Car {
    public LuxuryCar(Location location) {
        super(CarType.LUXURY, location);
    }
    @Override
    void construct() {
        System.out.println("connecting to Luxury Car");
    }
}
class MiniCar extends Car {
    public MiniCar(Location location) {
        super(CarType.MINI, location);
    }
    @Override
    void construct() {
        System.out.println("connecting to Mini Car");
    }
}
class MicroCar extends Car {
    public MicroCar(Location location) {
        super(CarType.MINI, location);
    }
    @Override
    void construct() {
        System.out.println("connecting to Micro Car");
    }
}
class IndianCarFactory {
    static Car buildCar(CarType carType) {
        Car car = null;
        switch (carType) {
            case MICRO:
                car = new MicroCar(Location.INDIA);
                break;
            case MINI:
                car = new MiniCar(Location.INDIA);
                break;
```

```
                case LUXURY:
                    car = new LuxuryCar(Location.INDIA);
                    break;
            }
            return car;
        }
    }
    class DefaultCarFactory {
        static Car buildCar(CarType carType) {
            Car car = null;
            switch (carType) {
                case MICRO:
                    car = new MicroCar(Location.AUS);
                    break;
                case MINI:
                    car = new MiniCar(Location.AUS);
                    break;
                case LUXURY:
                    car = new LuxuryCar(Location.AUS);
                    break;
            }
            return car;
        }
    }
    class USACarFactory {
        static Car buildCar(CarType carType) {
            Car car = null;
            switch (carType) {
                case MICRO:
                    car = new MicroCar(Location.USA);
                    break;
                case MINI:
                    car = new MiniCar(Location.USA);
                    break;
                case LUXURY:
                    car = new LuxuryCar(Location.USA);
                    break;
            }
            return car;
        }
    }
    class CarFactory {
        Car car = null;
        static Car buildCar(CarType carType, Location location) {
            Car car = null;
            switch (location) {
                case INDIA:
                    car = IndianCarFactory.buildCar(carType);
                    break;
                case USA:
                    car = USACarFactory.buildCar(carType);
                    break;
                case AUS:
                    car = DefaultCarFactory.buildCar(carType);
                    break;
            }
            return car;
        }
    }
    public class AbstractFactory3 {
        public static void main(String[] args) {
            System.out.println(CarFactory.buildCar(CarType.MICRO, Location.AUS));
            System.out.println(CarFactory.buildCar(CarType.MINI, Location.INDIA));
        }
```

```
}
```

4.

```
   class Student{
     private String name;
     private Integer id;
     private Integer fee;
     private Integer standard;
     private String address;
     private int batch;
     public Student(StudentBuilder studentBuilder) {
          id = studentBuilder.getId();
          name = studentBuilder.getName();
          fee = studentBuilder.getFee();
          standard = studentBuilder.getStandard();
          address = studentBuilder.getAddress();
          batch=studentBuilder.getBatch();
     }
     public String getName() {
          return name;
     }
     public void setName(String name) {
          this.name = name;
     }
     public Integer getId() {
          return id;
     }
     public void setId(Integer id) {
          this.id = id;
     }
     public Integer getFee() {
          return fee;
     }
     public void setFee(Integer fee) {
          this.fee = fee;
     }
     public int getStandard() {
          return standard;
     }
     public void setBatch(int batch){
          this.batch=batch;
     }
     public int getBatch(){
          return batch;
     }
     public void setAddress(String address){
          this.address=address;
     }
     public String getAddress() {
          return address;
     }
     @Override
     public String toString() {
          return "Employee{" +
                    "name='" + name + '\'' +
                    ", id=" + id +
                    ", fee=" + fee +
                    ", standard=" + standard +
                    ", address=" + address +
                    ", batch=" + batch +
                    '}';
     }
}
```

```
class StudentBuilder{
     private String name;
     private Integer id;
     private Integer fee;
     private String address;
     private Integer standard;
     private Integer batch;
     public StudentBuilder(String name, Integer id) {
          this.name = name;
          this.id = id;
     }
     public String getName() {
          return name;
     }
     public StudentBuilder setName(String name) {
          this.name = name;
          return this;
     }
     public Integer getId() {
          return id;
     }
     public StudentBuilder setId(Integer id) {
          this.id = id;
          return this;
     }
     public Integer getFee() {
          return fee;
     }
     public StudentBuilder withFee(Integer fee) {
          this.fee= fee;
          return this;
     }
     public StudentBuilder withStandard(Integer standard){
          this.standard=standard;
          return this;
     }
     public StudentBuilder withBatch(Integer batch){
          this.batch=batch;
          return this;
     }
     public StudentBuilder withAddress(String address){
          this.address=address;
          return this;
     }
     public int getStandard() {
          return standard;
     }
     public void setStandard(int standard){
          this.standard=standard;
     }
     public String getAddress(){
          return address;
     }
     public void setBatch(int batch){
          this.batch=batch;
     }
      public int getBatch(){
          return batch;
      }
      public Student build() {
          return new Student(this);
     }
}
public class Builder4 {
```

```
    public static void main(String[] args) {
        Student student = new StudentBuilder("Aditya",25)
                .withFee(11500)
                .withStandard(16)
                .withAddress("patna")
                .withBatch(2)
                .build();
        System.out.println(student);
    }
}


//5/

class Student5 {
    private final String firstName; // required
    private final String lastName; // required
    private final int age; // optional
    private final String phone; // optional
    private final String address; //optional
    private final int standard;// optional
    private Student5(StudentBuilder builder) {
        this.firstName = builder.firstName;
        this.lastName = builder.lastName;
        this.age = builder.age;
        this.phone = builder.phone;
        this.address = builder.address;
        this.standard = builder.standard;
    }
    //All getter, and NO setter to provde immutability
    public String getFirstName() {
        return firstName;
    }
    public String getLastName() {
        return lastName;
    }
    public int getAge() {
        return age;
    }
    public String getPhone() {
        return phone;
    }
    public String getAddress() {
        return address;
    }
    public int getStandard() {
        return standard;
    }
    @Override
    public String toString() {
        return "User: " + this.firstName + ", " + this.lastName + ", " + this.age + ", "
+ this.phone + ", " + this.address + ", " + standard;
    }
    public static class StudentBuilder {
        private final String firstName;
        private final String lastName;
        private int age;
        private String phone;
        private String address;
        private int standard;
        public StudentBuilder(String firstName, String lastName) {
            this.firstName = firstName;
            this.lastName = lastName;
        }
```

```
        public StudentBuilder age(int age) {
            this.age = age;
            return this;
        }
        public StudentBuilder phone(String phone) {
            this.phone = phone;
            return this;
        }
        public StudentBuilder address(String address) {
            this.address = address;
            return this;
        }
        public StudentBuilder standard(int standard) {
            this.standard = standard;
            return this;
        }
        public Student5 build() {
            Student5 student = new Student5(this);
            validateUserObject(student);
            return student;
        }
        private void validateUserObject(Student5 student) {
            //Do some basic validations to check
            //if user object does not break any assumption of system
            System.out.println("user validated");
        }
        public static void main(String[] args) {
            Student5 user1 = new Student5.StudentBuilder("Aditya", "Kumar")
                    .age(25)
                    .phone("7979765250")
                    .address("patna")
                    .build();
            System.out.println(user1);
            Student5 user2 = new Student5.StudentBuilder("vidit", "Gupta")
                    .age(24)
                    .phone("1234567")
                    .build();
            System.out.println(user2);
            Student5 user3 = new Student5.StudentBuilder("Rajnesh", "Ranjan")
                    .age(26)
                    //.phone("1234567")
                    //.address("Fake address 1234")
                    .build();
            System.out.println(user3);
        }
    }
}


//6

interface Size{
    String info();
}
class Pizza implements Size{
    private String size;
    public Pizza(String size) {
        this.size = size;
    }
    public String getSize() {
        return size;
    }
    public void setSize(String size) {
        this.size = size;
```

```
    }
    @Override
    public String info() {
        return "Pizza size  : " + size;
    }
}
class PizzaWithToopin implements Size {
    private Size shape;
    private String choice;
    public PizzaWithToopin(Size shape, String choice) {
        this.shape = shape;
        this.choice = choice;
    }
    public Size getShape() {
        return shape;
    }
    public void setShape(Size shape) {
        this.shape = shape;
    }
    public String getChoice() {
        return choice;
    }
    public void setChoice(String choice) {
        this.choice = choice;
    }
    @Override
    public String info() {
        return shape.info() + " with Toopin : " + choice;
    }
}
class Decorator6 {
    public static void main(String[] args) {
        Pizza square= new Pizza("medium");
        System.out.println(square.info());
        PizzaWithToopin toopin = new PizzaWithToopin(new Pizza("small"), "pottato");
       System.out.println(toopin.info());

    }
}

//7

import java.util.ArrayList;
import java.util.List;
interface Employee
{
    public void showEmployeeDetails();
}
class Development implements Employee
{
    private String name;
    private long empId;
    private String position;
    public Development(long empId, String name, String position)
    {
        this.empId = empId;
        this.name = name;
        this.position = position;
    }
    @Override
    public void showEmployeeDetails()
    {
        System.out.println(empId+" " +name+ " " + position );
    }
}
```

```
class Management implements Employee
{
     private String name;
     private long empId;
     private String position;
     public Management(long empId, String name, String position)
     {
          this.empId = empId;
          this.name = name;
          this.position = position;
     }
     @Override
     public void showEmployeeDetails()
     {
          System.out.println(empId+" " +name+ " " + position );
     }
}
class CompanyDirectory implements Employee
{
     private List<Employee> employeeList = new ArrayList<Employee>();
     @Override
     public void showEmployeeDetails()
     {
          for(Employee emp:employeeList)
          {
               emp.showEmployeeDetails();
          }
     }
     public void addEmployee(Employee emp)
     {
          employeeList.add(emp);
     }
     public void removeEmployee(Employee emp)
     {
          employeeList.remove(emp);
     }
}
class CompositeDesign7
{
     public static void main (String[] args)
     {
          Development dev1 = new Development(1, "Aditya kumar", "Trainee Developer");
          Development dev2 = new Development(2, "Rajnesh Ranjan", "Developer");
          CompanyDirectory engDirectory = new CompanyDirectory();
          engDirectory.addEmployee(dev1);
          engDirectory.addEmployee(dev2);
          Management man1 = new Management(3, "Radav Garg", "SEO Manager");
          Management man2 = new Management(4, "Ravish", "Senior HR");
          CompanyDirectory accDirectory = new CompanyDirectory();
          accDirectory.addEmployee(man1);
          accDirectory.addEmployee(man2);
          CompanyDirectory directory = new CompanyDirectory();
          directory.addEmployee(engDirectory);
          directory.addEmployee(accDirectory);
          directory.showEmployeeDetails();
     }
}
```

```java
//8

class NewStudent implements Accessible{
     private int age;
     private String name;
     public NewStudent(int age, String name) {
          this.age = age;
          this.name=name;
     }
     public int getAge() {
          return age;
     }
     public void setAge(int age) {
          this.age = age;
     }
     public String getName(){
          return name;
     }
     public void setName(String name){
          this.name=name;
     }
     public void access(){
          System.out.println("record access by student");
     }
}
interface Accessible{
     void access();
}
class Admin implements Accessible{
     protected Accessible accessible;
     public Admin(Accessible accessible) {
          this.accessible= accessible;
     }
     @Override
     public void access() {
          System.out.println("access the record");
     }
}
class AdminProxy  extends Admin{
     public AdminProxy(Accessible accessible) {
          super(accessible);
     }
     @Override
     public void access() {
          System.out.println("admin access the data by using proxy");
     }
}
class ProxyDesign8 {
     public static void main(String[] args) {
          NewStudent aditya= new NewStudent(25,"aditya");
          Accessible data= new AdminProxy(aditya);
          data.access();
     }
}
```