Programming Paradigms Tutorials
Object Oriented Programming in Scala

Object oriented programming

Object oriented programming way in Scala is very similar to Java.
Let's do some example with pizza.
Instead of enumerations we use sealed trait in Scala

```
sealed trait Topping
case object Cheese extends Topping
case object Pepperoni extends Topping
case object Sausage extends Topping
case object Mushrooms extends Topping
case object Onions extends Topping

sealed trait CrustSize
case object SmallCrustSize extends CrustSize
case object MediumCrustSize extends CrustSize
case object LargeCrustSize extends CrustSize

sealed trait CrustType
case object RegularCrustType extends CrustType
case object ThinCrustType extends CrustType
case object ThickCrustType extends CrustType
```

Given those enumerations, you can now start to create a few pizza-related classes for an order-entry system. First, here's a Pizza class:

```
class Pizza (
    var crustSize: CrustSize,
    var crustType: CrustType,
    var toppings: ArrayBuffer[Topping]
)
```

Next, here's an Order class, where an Order consists of a mutable list of pizzas and a Customer:

```
class Order (
    var pizzas: ArrayBuffer[Pizza],
    var customer: Customer
)
```

To define abstraction in Scala we should use traits. A trait is like an interface in Java with a partial implementation. In Scala, trait is a collection of abstract and non-abstract methods. You can create trait that can have all abstract methods or some abstract and some non-abstract methods. We can define trait which is responsible of adding and removing toppings:

```
Trait ToppingsManager {
    def addTopping(t: Topping)
    def removeTopping(t: Topping)
    def removeAllToppings()
}
```

And we can add this abstract definition to pour class:

```
class Pizza extends ToppingsManager (
    var crustSize: CrustSize,
    var crustType: CrustType,
    val toppings: ArrayBuffer[Topping]
) {

    def addTopping(t: Topping): Unit = toppings += t
    def removeTopping(t: Topping): Unit = toppings -= t
    def removeAllToppings(): Unit = toppings.clear()
}
```

Exercises

1. Provide implementation of Customer for Order class. Each customer should have name, phoneNumber and Address (another class)

2. Write a method which based on pizza details count a price for it.

3. Use trait to provide abstraction of animal, then use it to design class dog and presents some samples of usage.

4. Write a „Time” class in which the master constructor takes the time as an argument. The constructor should change negative values to 0. The class should have one field modifiable. Subsequent changes to the hour value are also intended to replace the values negative to 0.
   Add an accompanying object to the Time class to create instances of this class without using new.

5. Write singleton in Scala.

6. Write a class which represents book form the diagram below: