The Questions

1. (a) Implement a Counter class. The constructor should take an Int. The methods inc and dec should increment and decrement the counter respectively returning a new Counter. Here is an example of the usage:

scala> new Counter(10).inc.dec.inc.inc.count
res02: Int = 12

```
class Counter(var count: Int) {

  def inc() ={
    count += 1
    this
  }
  def dec() ={
    count -= 1
    this
  }
}
```

(b) Augment the Counter to allow the user can optionally pass an Int parameter to inc and dec. If the parameter is omitted it should default to 1.

```
class Counter(var count: Int) {

  def inc(x:Int=1) ={
    count += x
    this
  }
  def dec(x:Int=1) ={
    count -= x
    this
  }
}
```

(c) Re-implement Counter as a case class, using copy where appropriate. Additionally initialize count to a default value of 0.

```
case class Counter(var count: Int) {

  def inc(x:Int=1) ={
    count += x
    this
  }
  def dec(x:Int=1) ={
    count -= x
    this
  }
}
```

(d) Here is a simple class called Adder:

```
class Adder(amount: Int) {
        def add(in: Int) = in + amount
}
```

Extend Counter to add a method called adjust. This method should accept an Adder and return a new Counter with the result of applying the Adder to the count.

```
class Counter(var count: Int) {

  def inc(x:Int=1) ={
    count += x
    this
  }
  def dec(x:Int=1) ={
    count -= x
    this
  }
  def adjust(adder:Adder)={
    adder.add(count)
    this
  }

}
```

2. (a) Implement a companion object for a Person class containing an apply method that accepts a whole name as a single string rather than individual first and last names.

```
class Person(first:String, last:String) {

}


object Person{

  def apply(name:String)={
    val parts=name.split(" ")
    new Person(parts(0),parts(1))
  }
```

(b) What happens when we define a companion object for a case class?
Take our Person class and turn it into a case class. Make sure you still have the companion object with the alternate apply method as well.

When we define companion objects, we don't need to call constructor directly, we can use apply method and it does trick for us.

```
case class Person(first:String, last:String) {

}


object Person{

  def apply(name:String)={
    val parts=name.split(" ")
    new Person(parts(0),parts(1))
  }
```

3. (a) Write two classes, Director and Film, with fields and methods as follows:
Director should contain:
– a field firstName of type String
– a field lastName of type String
– a field yearOfBirth of type Int
– a method called name that accepts no parameters and returns the full name
Film should contain:
– a field name of type String
– a field yearOfRelease of type Int
– a field imdbRating of type Double
– a field director of type Director
– a method directorsAge that returns the age of the director at the time of release
– a method isDirectedBy that accepts a Director as a parameter and returns a Boolean

(b) Write companion objects for Director and Film as follows:
The Director companion object should contain:
– an apply method that accepts the same parameters as the constructor of the class and returns a new Director;
– a method older that accepts two Directors and returns the oldest of the two.

The Film companion object should contain:
– an apply method that accepts the same parameters as the constructor of the class and returns a new Film;
– a method highestRating that accepts two Films and returns the highest imdbRating of the two
– a method oldestDirectorAtTheTime that accepts two Films and returns the Director who was oldest at the respective time of filming.

```
class Director(val firstName: String, val lastName: String, val
yearOfBirth: Int) {
  def name() = {
    firstName + "  " + lastName
  }

}

object Director {
  def apply(firstName: String, lastName: String, yearOfBirth: Int) = {
    new Director(firstName, lastName, yearOfBirth)
  }
  def older(d1: Director, d2: Director) = {
    if (d1.yearOfBirth < d2.yearOfBirth) d1 else d2
  }
}
```

```
class Film(val name: String, val yearOfRelease: Int,
           val imdbRating: Double, val director: Director) {
  def directorsAge() = {
    director.yearOfBirth - yearOfRelease
  }
  def isDirectedBy(director: Director): Boolean = {
    (director.firstName == this.director.firstName &&
      director.lastName == this.director.lastName &&
      director.yearOfBirth == this.director.yearOfBirth)
  }

  def copy(newName: String = name, releaseYear: Int = yearOfRelease,
rating: Double = imdbRating,
           directorPerson: Director = director): Film =
    new Film(newName, releaseYear, rating, directorPerson)

}

object Film {
  def apply(name: String, yearOfRelease: Int, imdbRating: Double,
director: Director) = {
    new Film(name, yearOfRelease, imdbRating, director)
  }
  def oldestDirectorAtTheTime(f1: Film, f2: Film)  = {
    if (f1.directorsAge() > f2.directorsAge()) f1.director else
f2.director
  }
  def highestRating(f1: Film, f2: Film) = {
    if (f1.imdbRating > f2.imdbRating) f1.imdbRating else f2.imdbRating
  }

}
```

(c) We can dispose of much of the boilerplate by converting the Director and Film classes to case classes. Do this conversion and work out what code we can remove.

```
case class Director(val firstName: String, val lastName: String, val
yearOfBirth: Int) {
  def name() = {
    firstName + "  " + lastName
  }
}
object Director {
  def older(d1: Director, d2: Director) = {
    if (d1.yearOfBirth < d2.yearOfBirth) d1 else d2
  }
}
```

```
case class Film(val name: String, val yearOfRelease: Int,
           val imdbRating: Double, val director: Director) {
  def directorsAge() = {
    director.yearOfBirth - yearOfRelease
  }
  def isDirectedBy(director: Director): Boolean = {
    (director.firstName == this.director.firstName &&
      director.lastName == this.director.lastName &&
      director.yearOfBirth == this.director.yearOfBirth)
  }
}

object File {
  def oldestDirectorAtTheTime(f1: Film, f2: Film)  = {
    if (f1.directorsAge() > f2.directorsAge()) f1.director else
f2.director
  }
  def highestRating(f1: Film, f2: Film) = {
    if (f1.imdbRating > f2.imdbRating) f1.imdbRating else f2.imdbRating
  }

}
```