Scala会 中間テスト1 解答例

1. HelloWorldとそれをコンパイル・実行するコマンドを書け。

```
// どちらでも良い。
object Hello{
  def main(args: Array[String]) = {
    println("Hello World")
  }
}

object Hello extends App{
  println("Hello World" )
}
```

```
$ scalac ~~.scala

$ scala Hello
```

2. valとvarの違いは何か。また、オブジェクトの状態を変えるもののこと。変えないもののことをなんと呼ぶか。変えないもののオブジェクトの例を一つあげよ。

   valは再代入不可(値)、varは再代入可(変数)。状態を変えるものは、ミュータブル。変えないものをイミュータブル。JavaのStringや、ScalaのListなど。ただし、SetやHashなどはどちらも用意されているので、 scala.collection.immutable.Set のように答える必要がある。

3. Scalaでは、クラス定義のキーワード**class**の代わりに、**object**というキーワードを使うことがある。この`object`というキーワードは、何を表すものか。またJavaと比べて、どのような利点を持つかを説明せよ。

   シングルトンオブジェクト。Javaではクラス共通の値やメソッドを提供するときにstaticキーワードを使う。しかしstaticは静的に用意する仕組みなのでオブジェクト指向の考えにマッチしない(サブクラスの使用ができない。マルチスレッドプログラミングでの不整合。そもそもオブジェクトではない。)。しかしシングルトンオブジェクトは、シングルトンパターンのシンタックスシュガーなので、オブジェクト指向かつインスタンスが一つでしかないことを1キーワードで保証することができる。(シングルトンパターンの利点を調べてみよう。)

4. 同じソースで同じclass名、同じobject名を持つもの同士があるとき、そのobjectのことを何と呼ぶか。また、このobjectは呼び出しを省略することができる、あるメソッドを持つことができる。このメソッドの実用例を示せ。

   コンパニオンオブジェクト(逆にクラスの方はコンパニオンクラス)。applyメソッド。
   `Array(1,2,3)` `List(1,2,3)` などは、 `Array.apply(1,2,3)` `List.apply(1,2,3)`の省略形である。

5. Scalaでは、クラスにおいて、２つのコンストラクタを持つことができる。この２つのコンストラクタの名前、及び実用例を示せ。またコンストラクタがどのような制限を持つか答えよ。

   基本コンストラクタ(プライマリコンストラクタ)、補助コンストラクタ。

```
class Ratinal(n: Int, d: Int){
val number: Int = n
val denom: Int = d

def this(n: Int) = this(n, 1)

}
```

   最初に他のコンストラクタの呼び出しをしなければならない。つまり、基本コンストラクタが必ず呼ばれる必要がある。基本コンストラクタの代用ではなく、あくまで補助なので補助コンストラクタ。ちなみにあまり補助コンストラクタは用いられずapplyをでファクトリメソッドとして、使うケースが多い。

6. Scalaは純粋なオブジェクト指向である。これがわかる事柄をJavaと対比しつつ示せ。

   Javaではプリミティブ型が単なる値なのに対して、Scalaではプリミティブ型を含むすべての値はオブジェクトである。(Int, Double, Float 全て大文字から始まり、メソッドを持つ)
   全ての演算子はメソッド呼び出しである。 1 + 2 → 1.+(2) など。
   staticキーワードはなく、シングルトンオブジェクトを用いる(問3)。

7. Scalaでは、そのクラスが持っている以上に「リッチラッパー」により、メソッドを提供することができる。リッチラッパーは何という仕組みにより提供されているか答えよ。また基本型におけるリッチラッパーの実例をいくつか示せ。

implicit conversion(暗黙の型変換)。

```
1 max 2
0 min 3
3 round
4 to 6
```

など

8. `(1 to 100)` によって与えられる数列から偶数のみを取り出して、全ての要素を二倍した数列を得るプログラムを書け。ただし、高階関数を使った方法とfor式を使った方法の2種類で実装せよ。

```
(1 to 100).filter(_ % 2 == 0).map(_ * 2)

for(i <- (1 to 100) if i % 2 ==0) yield i * 2
```

9. 以下のコードを関数型スタイルのコードにリファクタリングせよ。

```
def printMultiTable() {
  var i = 1
  // ここではiだけがスコープに入っている
  while (i <= 10) {
    var j = 1
    // ここではiとjがスコープに入っている
    while (j <= 10) {
      val prod = (i * j).toString
      // ここではi、j、prodがスコープに入っている
      var k = prod.length
      // ここではi、j、prod、kがスコープに入っている
      while (k < 4) {
        print(" ")
        k += 1
      }
      print(prod)
      j += 1
    }
    // iとjはまだスコープに入っている。prodとkはスコープから外れている
    println()
    i += 1
  }
  // iはまだスコープに入っている。j、prod、kはスコープから外れている
}
```

基準は、println、varなどの副作用の排除。 値を返さない制御構文whileを排除。再利用がしやすいレベルで関数に分ける。for式や高階関数の適切な使用をする。などなど。

```
def makeRowSeq(row: Int): String = {
   for(col <- 1 to 10) yield {
     val prod = (row * col).toString
     val padding = " " * (4 - prod.length)
     padding + prod
   }
}

def makeRow(row: Int) = makeRowSeq(row).mkString

def multiTable() = {
   val tableSeq = for(row <- 1 to 10) yield makeRow(row)
   tableSeq.mkString( "\n")
}
```

10. 以下のコードは名前渡しパラメータという機能を使っている。このようなパターンのことを何と呼ぶか。また、以下のパターンを利用しRubyの times (指定した回数だけ、ブロックの処理を繰り返す構文) を再現せよ。なおInt型のメソッドとしてtimesを提供する必要はない。

```
def withPrintWriter(file: File, op: PrintWriter => Unit) {
  val writer = new PrintWriter(file)
  try {
    op(writer)
  } finally {
    writer.close()
  }
}

withPrintWriter(
  new File("date.txt"),
  writer => writer.println(new java.util.Date)
)
```

```
object Main extends App{

  def times(time: Int, block: => Unit): Unit = {
    def loop(n: Int):Unit = {
      if(n > 0){
        block
        loop(n - 1)
      }
    }
    loop(time)
  }

  times(10,
    {
      println("Hello")
    }
  )
}
```

↓こっちが書けたら + 10点。

```
1   object Helpers {
2     implicit class IntWithTimes(x: Int) {
3       def times[A](f: => A): Unit = {
4         def loop(current: Int): Unit =
5           if(current > 0) {
6             f
7             loop(current - 1)
8           }
9         loop(x)
10      }
11    }
12  }
```

```
1   import Helpers._
2
3   scala> 5 times println("HI")
```