object Assignment\_2 {  
 val days = List("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday")  
 val products = Map("fruits" -> 1000, "vegetables" -> 500)  
 def main(args: Array[String]): Unit = {  
 println("Task\_1:")  
 println(Task\_1("Monday"))  
 println(Task\_1("Saturday"))  
 println(Task\_1("8th day"))  
 println("Task\_2:")  
 val b: BankAccount = new BankAccount()  
 b.currentBalance  
 b.deposits(20.3)  
 b.currentBalance  
 b.withdraw(10)  
 b.currentBalance  
 new BankAccount(1000).currentBalance  
 println("Task\_3:")  
 val adam: PersonTask\_3 = new PersonTask\_3("Adam", "Sandler")  
 adam.greeting(new PersonTask\_3("Bakdaulet", "Tabynbaev"))  
 adam.greeting(new PersonTask\_3("Harley", "Davidson"))  
 adam.greeting(new PersonTask\_3("Bradley", "Cooper"))  
 println("Task\_4:")  
 println(Task\_4(x => x \* x, 2))  
 println("Task\_5:")  
 object p1 extends PersonTask\_5("Bakdaulet", "Tabynbaev") with Employee  
 p1.salary\_(1000)  
 p1.taxToPay  
 object p2 extends PersonTask\_5("Bakdaulet", "Tabynbaev") with Student  
 p2.taxToPay  
 object p3 extends PersonTask\_5("Bakdaulet", "Tabynbaev") with Teacher  
 p3.salary\_(1000)  
 p3.taxToPay  
 object p4 extends PersonTask\_5("Bakdaulet", "Tabynbaev") with Employee with Student  
 p4.salary\_(1000)  
 p4.taxToPay  
 object p5 extends PersonTask\_5("Bakdaulet", "Tabynbaev") with Student with Employee  
 p5.salary\_(1000)  
 p5.taxToPay  
 }  
 def Task\_1(day: String): String = day match {  
 case d if days.map(\_.toLowerCase()).filter(!\_.startsWith("s")).contains(d.toLowerCase()) => return "work"  
 case d if days.map(\_.toLowerCase()).filter(\_.startsWith("s")).contains(d.toLowerCase()) => return "weekends"  
 case \_ => return "no such day"  
 }  
  
 def Task\_4(f: Int => Int, a: Int): Int = f(f(f(a)))  
  
}  
  
class BankAccount(private var balance: Double) {  
  
 def this() { this(0) }  
  
 def deposits(amount: Double): Unit = {  
 if (amount > 0) balance = balance + amount  
 else throw new Exception("Incorrect input")  
 }  
  
 def withdraw(amount: Double): Unit = {  
 if (0 < amount && amount <= balance) {  
 balance = balance - amount  
 } else throw new Exception("Incorrect input")  
 }  
  
 def currentBalance: Unit = println("Current balance: " + balance)  
}  
  
case class PersonTask\_3(var firstName: String, var lastName: String) {  
 def greeting(person: PersonTask\_3): Unit = person match {  
 case PersonTask\_3("Harley", \_) => println("Hi! Harley")  
 case PersonTask\_3(\_, "Tabynbaev") => println("Hi! Mr. Tabynbaev")  
 case PersonTask\_3(fn, ln) => println(s"Hello! $fn $ln")  
 }  
}  
  
abstract class PersonTask\_5(private var firstName: String, private var lastName: String) {  
 def taxToPay: Unit  
}  
trait Employee extends PersonTask\_5 {  
 private var sal: Double = \_  
 def salary = sal  
 def salary\_(s: Double): Unit = sal = s  
 override def taxToPay: Unit = println("Your paying tax: " + sal \* 0.2)  
}  
trait Student extends PersonTask\_5 {  
 override def taxToPay: Unit = println("Your paying tax: 0")  
}  
trait Teacher extends Employee {  
 override def taxToPay: Unit = println("Your paying tax: " + salary \* 0.1)  
}