object Assignment\_9 {  
 def main(args: Array[String]): Unit = {  
 println("Task\_1:")  
 val Task\_1: Task\_1[String] = new Task\_1("Hello")  
 Task\_1.applyFunction(a => a + a)  
 println(Task\_1.getContent)  
  
 println("Task\_2:")  
 var no: No = new No()  
 println(no.isInstanceOf[Maybe[\_]])  
 var yes: Yes[String] = new Yes("a")  
 println(yes.isInstanceOf[Maybe[\_]])  
  
 println("Task\_3:")  
 val Task\_3no: Task\_3[No] = new Task\_3[No](new No())  
 Task\_3no.applyFunction(a => a)  
 println(Task\_3no.getContent)  
 val Task\_3yes: Task\_3[Yes[String]] = new Task\_3[Yes[String]](new Yes("Hello"))  
 Task\_3yes.applyFunction(a => new Yes(a.getContent + a.getContent))  
 println(Task\_3yes.getContent.getContent)  
  
 println("Task\_4:")  
 val Task\_4no: Task\_4[No] = new Task\_4[No](new No())  
 println(Task\_4no.getOrElse)  
 val Task\_4yes: Task\_4[Yes[String]] = new Task\_4[Yes[String]](new Yes("Hello"))  
 println(Task\_4yes.getOrElse)  
  
 }  
}  
class Task\_1[A](c: A) {  
 private var \_c: A = c  
 def getContent: A = \_c  
 def applyFunction(f: A => A): A =  
 {  
 \_c = f(\_c)  
 return \_c  
 }  
}  
  
trait Maybe[A]  
class No extends Maybe[Nothing]  
class Yes[A](value: A) extends Maybe[A] {  
 private var v: A = value  
 def getContent: A = v  
}  
  
class Task\_3[A](c: A) {  
 private var \_c: A = c  
 def getContent: A = \_c  
  
 def applyFunction(f: A => A): A =  
 {  
 if (f(\_c).isInstanceOf[No])  
 return \_c  
 else if (f(\_c).isInstanceOf[Yes[\_]]) {  
 \_c = f(\_c).asInstanceOf[A]  
 return \_c  
 } else  
 return null.asInstanceOf[A];  
 }  
}  
  
class Task\_4[A](c: A) {  
 private var \_c: A = c  
 def getContent: A = \_c  
 def getOrElse[B]: B =  
 {  
 if (\_c.isInstanceOf[No])  
 return "This is class NO with no content".asInstanceOf[B]  
 else if (\_c.isInstanceOf[Yes[\_]])  
 return \_c.asInstanceOf[Yes[A]].getContent.asInstanceOf[B]  
 else  
 return null.asInstanceOf[B];  
 }  
}