Jedi 0.0

1. Create a Scala project called Jedi.

2. Add four packages to the project: expression, value, context, and test.

3. Using the class diagrams as a guide, define the classes/traits Expression, Literal, Identifier, Value, Notification, Real, Integer, Chars, and Boole. Be sure to put things in their proper packages.

4. To make everything compile you will need to also define Environment. For now, something like this should work:

```
class Environment extends collection.mutable.HashMap[Identifier, Value]
```

5. Complete the implementations of execute for Literal and Identifier. Here's the start of Identifier:

```
case class Identifier(val name: String) extends Expression {
   override def toString = name
   def execute(env: Environment) = ???
}
```

Notes:

- All expression classes must be case classes. This will be required by our parsers. Scala automatically generates a companion object with an apply method for case classes.

6. A notification encapsulates some message such as "done" or "ok". Complete the implementation of the Notification class. Add a companion object with an apply method and pre-defined notifications OK, DONE, and UNSPECIFIED.

7. Complete the following implementation of the Integer class:

```
case class Integer(val value: Int) extends Literal with Ordered[Integer] with Equals {
  def +(other: Integer) = Integer(this.value + other.value)
  // *, -, /
  def unary_- = ??? // unary negation
  override def toString = value.toString
  def compare(other: Integer): Int = if (this.value < other.value) -1 else if
(other.value < this.value) 1 else 0
  override def canEqual(other: Any) =  other.isInstanceOf[Integer]
  override def equals(other: Any): Boolean =
    other match {
       case other: Integer => this.canEqual(other) && (other.value == this.value)
       case _ => false
    }
  override def hashCode = this.toString.##
}

object Integer {
  implicit def intToReal(n: Integer): Real = Real(n.value.toDouble)
}
```

Notes:

- The parser constructs expressions from strings. To do this, all expression subclasses should be case classes.
- Arithmetic operators are overloaded. (How should division by 0 be handled, option or exception?)
- a < b calls a.compare(b), which returns 1, 0, or -1
- Integers are values and so must implement equals and hashCode.

- Even though a companion object is automatically generated for a case class, we can continue the declaration of the companion.
- Implicit methods are automatically called to convert integers to reals.

8. Using the Integer class as a model, implement the Real class.

9. Implement the Boole class with binary && and || operators and a unary ! operator.

Notes:

- The name of a unary operator begins with the prefix "unary_". For example"

```
def unary_! = ...
```

- Each implementation should simply be a translation into the equivalent Scala expression. Remember, Jedi is the object language and Scala is the meta language.

10. Implement the Chars class. This class should support <, ==, substring, and +:

```
-> "cat" < "dog"
true
-> def animal = "cat" + "fish"
ok
-> animal
catfish
-> animal == "catfish"
true
```

11. Test your implementations using the following test files:

CharsTest.scala

NumberTest.scala

BooleTest.scala

ExpTest.scala