Java Interface

- Both Java and Scala support single inheritance: a class can only extend one other class.
- A Java class can implement any number of interfaces.

Java Interface - Continue

- A Java interface is similar to an abstract class in that they can not be instantiated, i.e. no interface objects or abstract class objects.
- Unlike abstract class which can declare abstract methods and implement concrete methods, an interface can only declare abstract methods, not instance variables, nor implemented methods.

Scala Trait

- A Scala class can extend only one other class but extend any number of traits.
- A trait can contain abstract and concrete methods.
- A trait can also contain abstract and concrete instance variables.
- class A extends Trait1 with Trait2
- class B extends A with Trait3 with Trait4
Trait Example

```scala
trait Logger {
  //abstract method
  def log(msg: String)
  //concrete methods
  def info(msg: String) {
    log("INFO: " + msg)
  }
  def warn(msg: String) {
    log("WARN: " + msg)
  }
  def severe(msg: String) {
    log("SEVERE: " + msg)
  }
}
```

class Account {
  protected var balance = 0.0
}

class SavingsAccount extends Account with Logger {
  var interest = 0.0;
  def withdraw(amount: Double) {
    if (amount > balance)
      severe("Insufficient funds")
    else
      balance -= amount
  }
  override def log(msg: String) {
    println(msg)
  }
}
```

Class Vs. Trait

- Extend a Class: all methods and instance variables of the extended class are inherited by the class.
- Extend a Trait: all methods and instance variables of the extended Trait are “mixed in” with the class, i.e. the compiler adds the mixed-in methods/instance variables to the class.

Mixed-In Trait

```scala
trait ShortLogger extends Logged {
  val maxLength = 15 ...
}
class SavingsAccount extends Account with ShortLogger {...}
```

Trait Construction Order

- Every trait has a single parameterless constructor.
- Constructors calling order:
  - The super class constructor is called first.
  - Trait constructors are executed next.
  - Traits are constructed from left-to-right
  - Within each trait, the parent traits get constructed first
  - If multiple traits share a common parent, the parent trait is constructed once only.
  - Finally, the sub-class constructor is called.
Example

```scala
class SavingsAccount extends Account with ConsoleLogger with ShortLogger
```

The constructors execute in the following order:
1. ?
2. ?
3. ?
4. ?
5. ?

Scala Code

```scala
abstract class Person {
  val id:Int //abstract
}
trait Student extends Person {
  override val id=99999;
  def learn() ="do homework"
}
trait Worker extends Person {
  override val id=99999;
  def work() = "design homework"
}
```

Simulate Multiple Inheritance

Scala Code

```scala
class WorkerStudent(sid:Int) extends Student with Worker {
  override val id=sid
  // Worker is mixin after Student. So, id refers to Worker id.
}
class StudentWorker(wid:Int) extends Worker with Student {
  override val id=wid
  // Student is mixin after Worker. So, schedule refers to Student schedule
}
```
Scala Code

```scala
object Test {
  def main(args: Array[String]) {
    var p = new Person(); // ?
    var s = new Student(); // ?
    var w = new Worker(); // ?
    var alice = new WorkerStudent(12);
    println("alice is a worker student:");
    println("id: "+ alice.id + "+ alice.learn" + alice.work);
    var bob = new StudentWorker(21);
    println("Bob is a student worker:");
    println("id: "+bob.id + "+bob.learn" + bob.work);
  }
}
```

Assignment 8

- Extend the class/trait hierarchy by adding
  - Volunteer trait
  - VolunteerWorker class
  - StudentVolunteer class
  - StudentWorkerVolunteer class