CURTIN UNIVERSITY (CRICOS number: 00301J)
Faculty of Engineering and Science
Department of Computing
Data Structures and Algorithms

Practical 1

Aims

- To begin implementing the system described in Practical 0 by building the basic classes for the DSAMining application.

Before the Practical:

- Read the requirements specification from the previous worksheet.

Activity 1: Implement Basic Classes

The DSAMining shipment system we will be building has a few fairly basic data-related classes that form the building blocks of the system. These include Ore, OrePile and ShipmentOrder. The UML class diagram in Figure 1 of this worksheet shows the constructors, methods and fields that should be implemented for these classes.

- Develop the three classes (Ore, OrePile, ShipmentOrder) and implement them in Java.

<u>Implementing the Classes:</u>

- It's a good idea to design and implement your classes in the following order:
  - Write down the member fields (variables)
  - Write down getters/setters (accessors/mutators) for those member fields
  - Write down the constructors, which should initialise *all* of the member fields
  - Write down other methods

- This way, it becomes obvious that you can use the setters as appropriate in the constructors, simplifying your code since the constructor can then let the setter do input validation for it (ie: then you code input validation *only once* – in the setter).

- However, continue to *organise* your classes in code as follows:
  - Member fields
  - Constructors
  - Getters/Setters (keep the getXYZ() with the setXYZ())
  - Other methods
- Getters and setters should become no-brainers to you. In fact, here is a recipe you can follow for almost any getter/setter:

```
public class Ore {
   private char oreType;
   private String units;

   ...
   public String getUnits() {
      return units;
   }
   public void setUnits(String inUnits) {
      if ( (inUnits == null) || (inUnits.equals("")) )    // Validation
         throw new IllegalArgumentException("Units must not be blank");
      units = inUnits;      // Assignment
   }
```

Name from field

Data type from field

NOTE: We do **NOT** want to return *copies* in the getter for this system (which is important for objects).

- Constructors should also become quite straightforward. Usually you consider three constructors immediately:
    - Default constructor (no parameters)
    - ‘Alternate’ (standard) constructor (one parameter for each member field)
    - Copy constructor (a single parameter: an object of the same type as the current class which is to be duplicated).

- Still, you need to decide whether you actually need them all (particularly the default constructor).

- The ‘alternate’ or ‘standard’ constructor (there’s no good name for it!) takes the parameters and uses them to initialise the member fields. Often it’s a case of simply assigning the member field to its associated parameter, but sometimes it’s a little different (eg: you have an array as a member field: the parameter would be maxSize of the array, and you initialise the member array to be maxSize in capacity).
    - Also, you should have a close look at the standard constructor to see if you really should be passing in parameters for *all* member fields, or if some member fields are really internal-only and shouldn’t be initialised by parameters.

- **<u>COMPILE EARLY AND OFTEN</u>** – this avoids errors piling up into a huge list.

Ore Notes:

For Ore, you will need to define an oreType field. Only two types of oreType are allowed: Iron or Nickel. One way to define this field is to make it a **char** and then define two constants that are the only valid values for that field. In Java, constants are marked as **static final** (ie: static=global and final="cannot be changed"), and are usually also public. For example:

```
public class Ore
{
   public static final char  ORETYPE_IRON = 'I';
   public static final char ORETYPE_NICKEL = 'N';

   private char oreType;
   private String units;

   public Ore(char inOreType, String inUnits) {
      if (inOreType != Ore.ORETYPE_IRON) && (inOreType != Ore.ORETYPE_NICKEL))
         throw new IllegalArgumentException("Invalid ore type provided");

      setUnits(inUnits);
      oreType = inOreType;
   }
   ... and the other methods
}
```

inUnits must be one of "t" (tonnes), "kg" (kilograms) or "g" (grams). Note that these must be of type String " " and not characters ' '. Notice also that the setter setUnits() is used in the constructor since it will do input validation for us. On the other hand, since we don't have a setOreType() we must validate in the constructor. Don't forget to perform import validation on *everything*! Strings must be checked for null as well as a .equals() check against invalid values such as blank (ie: "").

OrePile Notes:

calcMetalWeight() is a function that *calculates* the weight from the member fields `weight` and `grade`. Do not pass anything into calcMetalWeight – the member fields are all that is needed. In this way, it is like a getter, *but it is calculated from other fields rather than simply returning a member field*. Not a whole lot else to say here – just don't forget to include validation

ShipmentOrder Notes:

Note that when you create a new ShipmentOrder, you must assign the orderID to a unique value. The easiest way to do this is to maintain a counter variable that holds the nextOrderID. This nextOrderID must be shared across all instances of ShipmentOrder objects: in other words, it must be a *global* variable, not just a member variable. The way to do this is to mark the nextOrderID variable as **static**, which means "this variable is global to / shared across all objects of this class". Note that nextOrderID appears in the UML diagram as underlined, which indicates it is a static field. Also note that the first OrderID should be 1 and not 0 (see code below).

To use this global variable, whenever you construct a new ShipmentOrder have the constructor assigns its orderID = nextOrderID and then increment nextOrderID. This will ensure every ShipmentOrder is unique. For example:

```
public class ShipmentOrder
{
   private static int nextOrderID = 1;     // Global counter
   private int orderID;
   ...      // Other ShipmentOrder fields
   public ShipmentOrder(String inCustName, String inDestn, Ore inOre,
                        double inMetalWt, double inPrice) {
      orderID = nextOrderID;        // Set unique ID
      nextOrderID++;                // Increment global counter
      ...
   }
}
```

Remember that in setShippedOreWt() you must set isPending to *false*, indicating that the ore has been shipped, while in getShippedOreWt() you need to throw an Exception if isPending is true, since this would indicate that the ore has not been shipped yet. Don't forget to implement calcAverageGrade() and calcShipmentValue(), both similar to OrePile.calcMetalWeight() in that they are *calculated* from member fields and have no import parameters. Note that calcAverageGrade () cannot be calculated until setShippedOreWt has been set (ie: when the shipment ore weight has been calculated by DSAShipmentManager and the shipment has been sent off – in other words, if isPending is true, throw an exception in calcAverageGrade() saying that no ore has been shipped to calculate the grade of).

Note that you must *not* pass in the shipped ore weight or isPending flag: when an order is initially placed, isPending must be defaulted to **true**. Thus ShipmentOrder constructor is a mix of an alternate and default c'tor.

Miscellaneous Notes:

Note that methods inherited from the root class Object (such as equals()) have not been explicitly represented in the UML diagram – it is optional as to whether you implement them or not since this application does not specifically need them. For equals(), here is a template that will work for any object you have:

```
public class YourClassNameHere {
  public boolean equals(Object obj) {
    boolean bEquals = false;
    if (obj instanceof YourClassNameHere) {
       // Do comparison checks here
       bEquals = true;
    }
    return bEquals;
  }
}
```

Activity 2: Unit Testing

It's usually a good idea to test your classes individually after you initially write them. Otherwise you end up building a full system from untested components, and hit dozens of bugs. This is called *unit testing*, (covered in OOPD and Intro to Software Engineering) and involves creating a main() that acts as a test harness to test each method of a given class for correctness. If you don't do it now, you'll end up doing it the hard way later on anyway – ***TEST EARLY AND OFTEN!!!***.

Appendix 1 contains a comprehensive test harnesses for Ore, OrePile and ShipmentOrder classes. Use these as a guide to writing your own (These were provided for previous worksheet exercises and will most likely not work now!)

Submission Deliverable:

Your Ore, OrePile and ShipmentOrder classes are <u>due at the beginning of your next</u> <u>tutorial</u>. The classes developed here will also form the foundation for the whole system, which you will also submit when an extended version is completed.

SUBMIT ELECTRONICALLY VIA BLACKBOARD, under the *Assessments* section.

If you finish early, use the rest of the practical to start the next worksheet, because that will be due later on.

Appendix 1 Test harness'

```
// UnitTestOre.java
import java.io.*;
import io.*;


public class UnitTestOre
{
      public static void main(String args[])
      {
        int iNumPassed;
        int iNumTests;
        Ore ore;
        int iOreType;
        String sUnits;

        iNumPassed = 0;
        iNumTests = 0;

        // Test with normal conditions (shouldn't throw exceptions)

        System.out.println("\n");
        System.out.println("Testing Normal Conditions - Constructor");
        System.out.println("=======================================");

        try {
            iNumTests++;
            System.out.print("Testing creation of Ore: ");
            ore = new Ore(Ore.ORETYPE_IRON, "t");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing inOreType: ");
            iOreType = ore.getOreType();
            if (iOreType != Ore.ORETYPE_IRON)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing inUnits: ");
            sUnits = ore.getUnits();
            if (sUnits.equals("t") == false)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            System.out.println("\n");
            System.out.println("Testing Normal Conditions - Setters and Getters");
            System.out.println("===============================================");

            iNumTests++;
            System.out.print("Testing setUnits('g'): ");
            ore.setUnits("g");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing getUnits(): ");
            sUnits = ore.getUnits();
            if (sUnits.equals("g") == false)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

        } catch(Exception e) { System.out.println("FAILED"); }

        // Tests with error conditions (SHOULD throw exceptions)
        // Testing constructor's parameters of ore type and units.

        System.out.println("\n");
        System.out.println("Testing Error Conditions - Constructor");
        System.out.println("======================================");

        // Testing ORE TYPE
        // This test would also be redundant if they choose to use enums instead of
constants.

        try {
            iNumTests++;
            System.out.print("Testing constructor (inOreType=-1): ");
            ore = new Ore(-1, "t");
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
            System.out.print("Testing constructor (inOreType=33): ");
            ore = new Ore(33, "t");
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        // Testing UNITS
        try {
            iNumTests++;
            System.out.print("Testing constructor (inUnits='a'): ");
            ore = new Ore(Ore.ORETYPE_NICKEL, "a");
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
            System.out.print("Testing constructor (inUnits=''): ");
            ore = new Ore(Ore.ORETYPE_NICKEL, "");
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        // Testing setters and getters - note we don't have an Ore Type setter
        // so we can't test whether the getter is accurate.

        System.out.println("\n");
        System.out.println("Testing Error Conditions - Setters");
        System.out.println("==================================");

        // Testing UNITS
        try {
            iNumTests++;
            ore = new Ore(Ore.ORETYPE_NICKEL, "kg");
            System.out.print("Testing setUnits('') (units=''): ");
            ore.setUnits("");
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
            ore = new Ore(Ore.ORETYPE_NICKEL, "kg");
            System.out.print("Testing  setUnits('abc')  (units  not  equal  to  't',
'kg', 'g'): ");
            ore.setUnits("abc");
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        System.out.println("\n");
        System.out.println("Number PASSED: " + iNumPassed + "/" + iNumTests + " ("
+ (int)(100.0*(double)iNumPassed/(double)iNumTests) + "%)");
    }
}




// UnitTestShipmentOrder.java
import java.io.*;
import io.*;

public class UnitTestOrePile
{
      public static void main(String args[])
      {
        int iNumPassed;
        int iNumTests;
        OrePile orePile;
        Ore ore;
        Ore inOre;
        double dWeight;
        double dGrade;
        double dMetalWeight;
        iNumPassed = 0;
        iNumTests = 0;
        dWeight = 0.0;
        dGrade = 0.0;
        dMetalWeight = 0.0;

        // Assuming Ore works...
        ore = new Ore(Ore.ORETYPE_IRON, "t");

        // Test with normal conditions (shouldn't throw exceptions)

        System.out.println("\n");
        System.out.println("Testing Normal Conditions - Constructor");
        System.out.println("=======================================");

        try {
            iNumTests++;
            System.out.print("Testing creation of OrePile: ");
            orePile = new OrePile(ore, 200, 50);
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing inWeight: ");
            dWeight = orePile.getWeight();
            if (dWeight != 200)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing inGrade: ");
            dGrade = orePile.getGrade();
            if (dGrade != 50)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing inOre: ");
            if  (orePile.getOre()  ==  null)     //  Would  need  .equals()  to  test
properly, but not req'd func
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            System.out.println("\n");
            System.out.println("Testing Normal Conditions - Setters and Getters");
            System.out.println("===============================================");

            iNumTests++;
            System.out.print("Testing setWeight(150): ");
            orePile.setWeight(150);
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing getWeight(): ");
            dWeight = orePile.getWeight();
            if (dWeight != 150)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing setGrade(65): ");
            orePile.setGrade(65);
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing getGrade(): ");
            dGrade = orePile.getGrade();
            if (dGrade != 65)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

        } catch(Exception e) { System.out.println("FAILED"); }


        // Tests with error conditions (SHOULD throw exceptions)
        // Testing constructor's parameters of weight and grade.

        System.out.println("\n");
        System.out.println("Testing Error Conditions - Constructor");
        System.out.println("======================================");

        //Testing WEIGHT
        try {
            iNumTests++;
            System.out.print("Testing constructor (inWeight=0): ");
            orePile = new OrePile(ore, 0, 50);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
            System.out.print("Testing constructor (inWeight<0): ");
            orePile = new OrePile(ore, -1, 50);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        // Testing GRADE
        try {
            iNumTests++;
            System.out.print("Testing constructor (inGrade=0): ");
            orePile = new OrePile(ore, 200, 0);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
            System.out.print("Testing constructor (inGrade>100): ");
            orePile = new OrePile(ore, 200, 101);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
            System.out.print("Testing constructor (inGrade<0): ");
            orePile = new OrePile(ore, 200, -1);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }


        // Testing setters and getters

        System.out.println("\n");
        System.out.println("Testing Error Conditions - Setters");
        System.out.println("==================================");

        orePile = new OrePile(ore, 200, 45);

        // Testing WEIGHT
        try {
            iNumTests++;
            System.out.print("Testing setWeight(0) (weight=0): ");
            orePile.setWeight(0);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
            System.out.print("Testing setWeight(-5) (weight<0): ");
            orePile.setWeight(-5);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        // Testing GRADE
        try {
            iNumTests++;
            System.out.print("Testing setGrade(0) (grade=0): ");
            orePile.setGrade(0);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
```

```
            System.out.print("Testing setGrade(101) (grade>100): ");
            orePile.setGrade(101);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
            System.out.print("Testing setGrade(-1) (grade<0): ");
            orePile.setGrade(-1);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }


        // Test metal weight calculation with normal conditions
        // (shouldn't throw exceptions)

        System.out.println("\n");
        System.out.println("Testing Other Methods");
        System.out.println("=====================");

        try {
            iNumTests++;
            System.out.print("Testing calcMetalWeight(): ");
            orePile = new OrePile(ore, 200, 50);
            dMetalWeight = orePile.calcMetalWeight();
            if (dMetalWeight == 100)
            {
                iNumPassed++;
                System.out.println("passed");
            }
            else
                throw new IllegalArgumentException("FAILED");
        } catch(Exception e) { System.out.println("FAILED"); }

        System.out.println("\n");
        System.out.println("Number PASSED: " + iNumPassed + "/" + iNumTests + " ("
+ (int)(100.0*(double)iNumPassed/(double)iNumTests) + "%)");
    }
}

//UnitTestShipmentOrder.java
import java.io.*;
import io.*;

public class UnitTestShipmentOrder
{

      public static void main(String args[])
      {
        int iNumPassed;
        int iNumTests;
        ShipmentOrder shipOrder;
        Ore ore;
```

```
OrePile orePile;
int iOrderID;
boolean bIsPending;
String sCustName;
String sShipDest;
double dOrderedMetalWt;
double dUnitPrice;
double dAveGrade;
double dShipValue;

iNumPassed = 0;
iNumTests = 0;
iOrderID = 0;
bIsPending = false;
sCustName = "";
sShipDest = "";
dOrderedMetalWt = 0.0;
dUnitPrice = 0.0;
dAveGrade = 0.0;
dShipValue = 0.0;

// Assuming Ore and OrePile work...
ore = new Ore(Ore.ORETYPE_IRON, "t");
orePile = new OrePile(ore, 500, 50);

// Test with normal conditions (shouldn't throw exceptions)

System.out.println("\n");
System.out.println("Testing Normal Conditions");
System.out.println("=========================");

try {
    iNumTests++;
    System.out.print("Testing creation of ShipmentOrder: ");
    shipOrder = new ShipmentOrder("John Smith", "Exmouth", ore, 400, 250);
    iNumPassed++;
    System.out.println("passed");

    iNumTests++;
    System.out.print("Testing OrderID: ");
    iOrderID = shipOrder.getOrderID();
    if (iOrderID != 1)
        throw new IllegalArgumentException("FAILED");
    iNumPassed++;
    System.out.println("passed");

    iNumTests++;
    System.out.print("Testing isPending: ");
    bIsPending = shipOrder.getIsPending();
    if (bIsPending == false)
        throw new IllegalArgumentException("FAILED");
    iNumPassed++;
    System.out.println("passed");
```

```
            iNumTests++;
            System.out.print("Testing inCustName: ");
            sCustName = shipOrder.getCustomerName();
            if (sCustName.equals("John Smith") == false)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing inShipDest: ");
            sShipDest = shipOrder.getShippingDest();
            if (sShipDest.equals("Exmouth") == false)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing inOre: ");
            if  (shipOrder.getOre()  ==  null)     //  Would  need  .equals()  to  test
properly, but not req'd func
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing inOrderedMetalWt: ");
            dOrderedMetalWt = shipOrder.getOrderedMetalWt();
            if (dOrderedMetalWt != 400)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing inUnitPrice: ");
            dUnitPrice = shipOrder.getUnitPrice();
            if (dUnitPrice != 250)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            System.out.println("\n");
            System.out.println("Testing Normal Conditions - Setters and Getters");
            System.out.println("===============================================");

            iNumTests++;
            System.out.print("Testing setShipDest(\"Broome\"): ");
            shipOrder.setShippingDest("Broome");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing getShipDest(): ");
            sShipDest = shipOrder.getShippingDest();
            if (sShipDest.equals("Broome") == false)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing setOrderedMetalWt(350): ");
            shipOrder.setOrderedMetalWt(350);
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing getOrderedMetalWt(): ");
            dOrderedMetalWt = shipOrder.getOrderedMetalWt();
            if (dOrderedMetalWt != 350)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing setUnitPrice(125): ");
            shipOrder.setUnitPrice(125);
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing getUnitPrice(): ");
            dUnitPrice = shipOrder.getUnitPrice();
            if (dUnitPrice != 125)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

        } catch(Exception e) { System.out.println("FAILED"); }

        // Testing getShippedOreWt() - should throw an exception at this point, so
catch it = passed
        try {
            iNumTests++;
            System.out.print("Testing getShippedOreWt(): ");
            shipOrder = new ShipmentOrder("John Smith", "Exmouth", ore, 400, 250);
            shipOrder.getShippedOreWt();
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }


        // Tests with error conditions (SHOULD throw exceptions)
        // Testing constructor's parameters of customer name,
        // shipping destination, ordered metal weight, unit price.
        // Assume ore is OK.

        System.out.println("\n");
        System.out.println("Testing Error Conditions - Constructor");
        System.out.println("======================================");

        //Testing CUSTOMER NAME
        try {
            iNumTests++;
            System.out.print("Testing constructor (inCustomerName=\"\"): ");
            shipOrder = new ShipmentOrder("", "Exmouth", ore, 200, 250);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        //Testing SHIPPING DESTINATION
        try {
            iNumTests++;
            System.out.print("Testing constructor (inShippingDest=\"\"): ");
            shipOrder = new ShipmentOrder("John Smith", "", ore, 200, 250);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        // Testing ORDERED METAL WEIGHT
        try {
            iNumTests++;
            System.out.print("Testing constructor (inOrderedMetalWt=0): ");
            shipOrder = new ShipmentOrder("John Smith", "Exmouth", ore, 0, 250);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
            System.out.print("Testing constructor (inOrderedMetalWt<0): ");
            shipOrder = new ShipmentOrder("John Smith", "Exmouth", ore, -1, 250);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        // Testing UNIT PRICE
        try {
            iNumTests++;
            System.out.print("Testing constructor (inUnitPrice=0): ");
            shipOrder = new ShipmentOrder("John Smith", "Exmouth", ore, 200, 0);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
            System.out.print("Testing constructor (inUnitPrice<0): ");
            shipOrder = new ShipmentOrder("John Smith", "Exmouth", ore, 200, -1);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        // Testing setters and getters

        System.out.println("\n");
        System.out.println("Testing Error Conditions - Setters");
        System.out.println("==================================");

        shipOrder = new ShipmentOrder("John Smith", "Exmouth", ore, 200, 250);

        // Testing CUSTOMER NAME
        try {
            iNumTests++;
            System.out.print("Testing setCustomerName(\"\"): ");
            shipOrder.setCustomerName("");
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        // Testing SHIPPING DESTINATION
        try {
            iNumTests++;
            System.out.print("Testing setShippingDest(\"\"): ");
            shipOrder.setShippingDest("");
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        // Testing ORDERED METAL WEIGHT
        try {
            iNumTests++;
            System.out.print("Testing setOrderedMetalWt(0): ");
            shipOrder.setOrderedMetalWt(0);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
            System.out.print("Testing setOrderedMetalWt(-1): ");
            shipOrder.setOrderedMetalWt(-1);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        // Testing UNIT PRICE
        try {
            iNumTests++;
            System.out.print("Testing setUnitPrice(0): ");
            shipOrder.setUnitPrice(0);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        try {
            iNumTests++;
            System.out.print("Testing setUnitPrice(-1): ");
            shipOrder.setUnitPrice(-1);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        // Testing SHIPPED ORE WEIGHT
        try {
            iNumTests++;
```

```
            System.out.print("Testing setShippedOreWt(-1): ");
            shipOrder.setShippedOreWt(-1);
            System.out.println("FAILED");
        } catch(Exception e) { iNumPassed++; System.out.println("passed"); }

        // Testing other methods.

        // Have valid setShippedOreWt(), test getShippedOreWt() to work without 
exceptions
        // Does getIsPending then start returning false. If not, then FAILED
        try {
            iNumTests++;
            System.out.print("Testing setShippedOreWt(300): ");
            shipOrder.setShippedOreWt(300);
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing getShippedOreWt(): ");
            if (shipOrder.getShippedOreWt() != 300)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");

            iNumTests++;
            System.out.print("Testing getIsPending(): ");
            if (shipOrder.getIsPending() == true)
                throw new IllegalArgumentException("FAILED");
            iNumPassed++;
            System.out.println("passed");
        } catch(Exception e) { System.out.println("FAILED"); }

        // Test average grade and shipment value calculations with normal 
conditions
        // (shouldn't throw exceptions).
        try {
            iNumTests++;
            System.out.print("Testing average grade calculation: ");
            shipOrder.setShippedOreWt(400);
            dAveGrade = shipOrder.calcAverageGrade();
            if (dAveGrade == 50)
            {
                iNumPassed++;
                System.out.println("passed");
            }
            else
                throw new IllegalArgumentException("FAILED");
        } catch(Exception e) { System.out.println("FAILED"); }

        try {
            iNumTests++;
            System.out.print("Testing shipment value calculation: ");
            dShipValue = shipOrder.calcShipmentValue();
            if (dShipValue == 50000)
            {
                iNumPassed++;
                System.out.println("passed");
            }
            else
                throw new IllegalArgumentException("FAILED");
        } catch(Exception e) { System.out.println("FAILED"); }

        System.out.println("\n");
        System.out.println("Number PASSED: " + iNumPassed + "/" + iNumTests + " ("
+ (int)(100.0*(double)iNumPassed/(double)iNumTests) + "%)");

      }
}
```