CS1302
Exam01 Review

PLAS:

Xander Aviles and (Aiden) Geonhui Lee

To be used in supplement with:

$ git clone git@github.com:xanaviles/cs1302-exam01Review.git

Navigating Unix

1. Change into cs1302-exam01Review and do not leave unless directed otherwise (cd)
2. Look around the directory (ignore the jar file for now) (ls and find)
3. Create src/cs1302/practice and src/cs1302/inter in one command and without changing directories (mkdir and pipes)
4. Move Moveable.java into src/cs1302/inter and move the rest of the java files into src/cs1302/practice (mv)
5. Create *cs1302.practice.Cow* by copying Bear.java to the new file, Cow.java (mv & cp)
6. [bonus] Change into the src/cs1302/practice directory, how would you change back into cs1302-exam01Review in one command? (using "..")

Directory Result:

cs1302-exam01Review
- listadt.jar // ignore for now
- misc // ignore this dir
- bin // don't forget to make bin
- src
    - cs1302
        - inter
            - Moveable.java
        - practice
            - Animal.java
            - Mammal.java
            - Fish. java
            - Cow.java
            - Bear.java

- **Commands not mentioned but should be reviewed:**

  grep, wc, echo, chmod (permissions, octal, ...), which, stat, pwd

Dependencies & UML

(Interfaces and Abstract Classes | Reference and Object Types)

1. Use cat to determine dependencies of the classes.

File Name	Type of Class	Depends On
Moveable.java	Interface	
Animal.java	Abstract	Moveable
Mammal.java	Abstract	Moveable, Animal
Cow.java	Concrete	Moveable, Animal, Mammal
Bear.java	Concrete	Moveable, Animal, Mammal
Fish.java	Concrete	Moveable, Animal

2. Examine the classes and draw the UML for these relationships.
   answer can be found here
   *remember you implement Interfaces and extend Abstract classes
3. Adjust your Cow.java variables and methods.
4. Compile these classes in the correct order.
5. Create the class and package for cs1302.drivers.UmlDriver

6. Paste this code into UmlDriver.java and examine the code.

   Before you compile and run it, predict what will happen by using the UML diagram to help

```
package cs1302.drivers;

import cs1302.inter.Moveable;
import cs1302.practice.Fish;
import cs1302.practice.Animal;
import cs1302.practice.Mammal;
import cs1302.practice.Cow;
import cs1302.practice.Bear;

public class UmlDriver {

   public static void main(String args[]) {
      // ---------------------------------------
      // refType varName = new objectType();
      // Which of these will compile?
      // ---------------------------------------

      Moveable m1 = new Fish("Goldfish");
      Moveable m2 = new Moveable();
      Moveable m3 = new Animal("Animal");


      Animal a1 = new Animal("Animal");
      Animal a2 = new Fish("Goldfish");
      Animal a3 = new Mammal("Mam");
      Animal a4 = new Cow("Moo");


      Mammal mam1 = new Fish("Fish");
      Mammal mam2 = new Bear("Brad");
      Bear b1 = new Bear("Wojtek");
      Bear b2 = new Cow("not a cow");

      // ---------------------------------------
      // let's test some methods !
      // which ones will compile?
      // ---------------------------------------

      // Moveable m1 = new Fish("Goldfish");
```

```
        System.out.println("----------  Moveable m1 = new
Fish(\"Goldfish\")");
        System.out.println("m1.canMove: " + m1.canMove);
        System.out.println("m1.hasLegs(): " + m1.hasLegs());
        m1.sound();


        // Animal a4 = new Cow("Moo");
        System.out.println("\n----------  Animal a4 = new Cow(\"Moo\")");
        System.out.println(a4.isFarmMammal());
        System.out.println("a4.hasLegs(): " + a4.hasLegs());
        a4.sound();


        // Animal a2 = new Fish("Goldfish");
        System.out.println("\n--------- Animal a2 = new Fish(\"Goldfish\")");
        System.out.println("a2.hasLegs(): " + a2.hasLegs());
        a2.sound();


        // Mammal mam2 = new Bear("Brad");
        System.out.println("\n--------- Mammal mam2 = new
Bear(\"Brad\")");
        System.out.println("mam2.hasLegs(): " + mam2.hasLegs());
        mam2.sound();
        mam2.setName("Brad!");
        System.out.println("mam2.setName(\"Brad!\") and
mam2.getName(): " + mam2.getName());
        System.out.println("mam2.canMove: " + mam2.canMove);


    } // main

} // umlDriver
```

Exception Handling

1. Compare the possible implementations for setName(String n)

Animal.java (fig. 1)

```
public void setName(String n) {
    if (n == null) {
        throw new NullPointerException("name cannot be null");
    } else if (n.isEmpty()) {
        throw new IllegalArgumentException("name cannot be empty string");
   } else {
        this.name = n;
   } // if

   System.out.println("setName is done");
} // setName
```

Animal.java (fig. 2)

```
public void setName(String n) {
    try {
        this.name = n;
    } catch (NullPointerException npe) {
        System.out.println("name cannot be null");
    } catch (IllegalArgumentException iae) {
        System.out.println(iae);
    } finally {
        System.out.println("setName is done");
    }
} // setName
```

Animal.java (fig. 3)

```
public void setName(String n) throws NullPointerException,
                                     IllegalArgumentException {
   this.name = n;

   System.out.println("setName is done");
}
```

Checked & Unchecked

1. Checked Exceptions:
    - you must import them
    - are “checked” at compile time
2. Unchecked Exceptions:
    - subclass of java.lang.RuntimeException
    - are “not checked” at compile time; you can compile without handling the exception
    - the exception is in the java.lang package

*not a comprehensive list; just examples

Checked Exceptions	Unchecked Exceptions
java.io.IOException java.io.FileNotFoundException	java.lang.NullPointerException java.lang.IllegalArgumentException java.lang.ArrayIndexOutOfBoundsException java.lang.IndexOutOfBoundsException

Javadoc

- Most common tag syntax (do not add the parentheses to your javadoc)

```
@throws (exceptionClassName) (description of why it is thrown)

@return (description of what is returned)

@param (parameterName) (description of parameter)
```

find a full list of tags here

- Javadoc Example

Animal.java

```
/**
   * This is a javadoc comment. Talk about what your method does here.
   * @return a string representation of the animal's name
   */
public String getName() {
   return this.name;
}

/**
   * This is a javadoc comment. Talk about your method here.
   * @throws NullPointerException if the specified string is null
   * @throws IllegalArgumentException if {@code n} is an empty string
   * @param n the name of the animal
   */
public void setName(String n) {
     try {
         this.name = n;
    } catch (NullPointerException npe) {
         System.out.println("name cannot be null");
    } catch (IllegalArgumentException iae) {
         System.out.println(iae);
    }
```

```
}
```

Nodes

1. Create cs1302.drivers.NodeDriver

```
package cs1302.drivers;

import cs1302.listadt.StringList;  // importing the jar file

public class NodeDriver {

	//be aware of 3 Node constructors

	// 1. Node()
	// 2. Node(String str)
	// 3. Node(String str, Node next)

	//Block 1
	 StringList.Node a = new StringList.Node();
	a.setName("1");

	//Block 2
	a.setNext(new StringList.Node("2"));
	a.getNext().setNext(new StringList.Node("3"));

	//Block 3
	StringList.Node b = new StringList.Node("0", a);

	//Block 4
	b.getNext().getNext().setNext("5");

	//Block 5
	b.getNext().setNext(b.getNext().getNext().getNext());

} // NodeDriver
```