Metropolitan State University, St. Paul, MN
ICS 372 Object-Oriented Design and Implementation
Class Exercise 2 Solution

1. Consider the following code.

```
public class EX1 {
   public void m1(int x) {
      int a;
      a = x;
      int b = a + 2;
      m2(b);
   }
   public int m2(int y) {
      if (y <= 2) {
         throw new MyException(y + " has a bad value");
      }
      int x = (int) (Math.random() * 100);
      return x + y;
   }
   public static void main(String[] args) {

      EX1 e1 = new EX1();
      e1.m1(5);
      e1.m1(-4);
   }
}

class MyException extends Exception {
   public MyException(String s) {
       super(s);
   }
}
```

   (a) The above code does not compile. Where is the problem?
   The method `m1()` is problematic because the method `m2()` throws an exception. The method `m2()` should either catch it or declare it as throwing it back to its caller.

   (b) If the method is not `m2(b)` invoked, would the code still have an error?
   Yes. The problem is with `m2()`. Its declaration must be fixed.

   (c) Come up with two different ways of getting rid of the syntax error.
   Approach 1: throw the exception back to the caller.

```
public class EX1 {
   public void m1(int x) throws MyException {
      int a;
```

```
      a = x;
      int b = a + 2;
      m2(b);
   }

    public int m2(int y) throws MyException {
      if (y <= 2) {
         throw new MyException(y + " has a bad value");
      }
      int x = (int) (Math.random() * 100);
      return x + y;
    }

    public static void main(String[] args) throws MyException {
         EX1 e1 = new EX1();
         e1.m1(5);
         e1.m1(-4);
     }
}
class MyException extends Exception {
     public MyException(String s) {
         super(s);
     }
}
```

Approach 2: Catch the exception. The rest of the code can remain the same.

In the previous approach, have `m2()` throws the exception and the code in `m1()` catch the exception.

```
public void m1(int x) {
   int a;
   a = x;
       int b = a + 2;
   try {
       m2(b);
   } catch (MyException me) {

   }
}
```

(d) Which calls from `main()` result in an exception?

The call `e1.m1(-4);`. This causes b to be -2 and this value results in the exception being thrown in `m2()`.

Consider the following code to implement an account for a bank.

```
public abstract class Account {
    private double balance;
```

```
    private int accountNumber;
    private static int counter = 1;

    public Account(double balance) {
        this.balance = balance;
        this.accountNumber = computeAccountNumber();
    }

    public int getAccountNumber() {
        return accountNumber;
    }

    private int computeAccountNumber() {
        return counter++;
    }

    public double getBalance() {
        return balance;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) {
        balance -= amount;
    }
}
```

2. Implement a method named `transferFrom()` with two parameters: `account` and `amount` to transfer `amount` dollars from `account` to `this` account. If there is enough balance, the money is transferred and the method returns `true`. Otherwise, it returns `false`.

```
public boolean transferFrom(Account account, double amount) {
   if (account.getBalance() >= amount) {
      account.withdraw(amount);
      this.deposit(amount);
      return true;
   }
   return false;
}
```

3. Implement the `equals()` and `hashCode()` methods correctly for the `Account` class. Two `Account` objects re equal if and only if they have the same account number.

```
@Override
public int hashCode() {
   return 71 * accountNumber;
}

@Override
public boolean equals(Object object) {
   if (this == object) {
      return true;
   }
   if (object == null) {
      return false;
   }
   if (!(object instanceof Account)) {
      return false;
   }
   Account other = (Account) object;
   if (accountNumber != other.accountNumber) {
      return false;
   }
   return true;
}
```

4. Create a class called `CheckingAccount` that inherits from `Account`. Every checking account has to maintain a minimum balance, which is the same for all checking accounts. It should be possible to change this minimum balance from time to time.

```
public class CheckingAccount extends Account {
    private static double minimumBalance;
    public CheckingAccount(double balance) throws Exception {
       super(balance);
       if (balance < minimumBalance) {
          throw new Exception("Need to maintain a minimum balance");
       }
    }

    public static double getMinimumBalance() {
       return minimumBalance;
    }

    public static void setMinimumBalance(double minimumBalance) {
       CheckingAccount.minimumBalance = minimumBalance;
    }

    @Override
    public void withdraw(double amount) {
       if (getBalance() >= minimumBalance) {
```

```
            super.withdraw(amount);
        }
    }

    @Override
    public boolean transferFrom(Account account, double amount) {
        if (account instanceof CheckingAccount) {
            if (account.getBalance() >= minimumBalance) {
                return super.transferFrom(account, amount);
            }
            return false;
        } else {
            return super.transferFrom(account, amount);
        }
    }

}
```

5. This question is based on the following classes.

```
public class Building {
protected int area;
}

public class CommercialBuilding extends Building {
}

public class ResidentialBuilding extends Building {
}

Object o;
Building b = new Building();
ResidentialBuilding r = new ResidentialBuilding();
CommercialBuilding c = new CommercialBuilding();
```

The first column in the following table shows statements using the above references. In the second column, write whether the error has a syntax error. If not, in the third column, write whether the execution of the statement is always without error or could cause an error on occasions.

Statement	Syntax Error?	Execution Always Correct or Crashes sometimes?
b = r;	No	Always correct
r = b;	Yes	NA
b = (Building) c;	No	Always correct
c = (CommercialBuilding) b;	No	Crashes sometimes
r = c;	Yes	NA
c = (CommercialBuilding) r;	Yes	NA
o = r;	No	Always correct
r = o;	Yes	NA
o = (Object) r;	No	Always correct
r = (ResidentialBuilding) o;	No	Crashes sometimes

6. Consider the following class definitions:

```
abstract class Animal {
public abstract void method1();
}

class Dog extends Animal {
public void method1() {
System.out.println("ruff ruff");
}
}

class Cat extends Animal {
public void method1() {
System.out.println("meow meow");
}
}

class Cow extends Animal {
public void method1() {
System.out.println("moo moo");
}

public void method2() {
System.out.println("I am brown");
}
}
```

For each row in the table below, check corresponding column of whether the statement is correct, generates compilation error, or generates run time error. If the statement is correct, show the output, and in the case of error, write the cause of the error.

Statement	Correct or Compilation error or Run-time error	Output or Cause of error (if any)
Animal a = new Animal();	Compilation error	Can't instantiate an abstract class
Animal d = new Dog(); d.method1();	Correct	ruff ruff
Animal w = new Cow(); w.method2();	Compilation error	There is no method named method2() in Animal
Animal c = new Cat(); ((Cow)c).method2();	No error	Runtime error; Can't cast Cat to Cow
Animal c = new Cat(); ((Cat)c).method2();	No error	Runtime error; No method named method2() in Cat