Objects Part 1.
This chapter was mostly dealt with objects except questions similar to these.

1. Create accessors and mutators for the following class. Make sure to check for valid values

```java
public class Person
{
    private String name;//Must not be null
    private int age;//Must be greater and including 0
    //Put accessors and mutators here
    //Accessors
    public String getName()
    {
        return this.name;
    }
    public int getAge()
    {
        return this.age;
    }
    //Mutators
    public void setName(String aName)
    {
        if(aName == null)
        {
            this.name = “none”;
        }
        else
        {
            this.name = aName;
        }
    }
    public void setAge(int anAge)
    {
        if(anAge >= 0)
        {
            this.age = anAge;
        }
        else
        {
            this.age = 0;
        }
    }
}
2. Write a method that converts the instance variable “feet” to a measurement in meters. The conversion between feet and meters is 1ft = 0.3048m

```java
public class Measurement {
    private double feet;
    // put the method here
    public double inMeters() {
        return this.feet * 0.3048;
    }
}
```

3. Write a method that adds all the elements of an instance variable array together and then prints the result.

```java
public class ArrayOfStuff {
    private int[] array = {1, 2, 3, 4, 5};
    // Put the method here
    public void printArray() {
        int result = 0;
        for (int i = 0; i < array.length; i++) {
            result += array[i];
        }
        System.out.println("The sum of the array is "+result);
    }
}
```
Objects Part 2.
This chapter further expanded the concept of objects by introducing Constructors, Static methods, and Overloading methods.

1. Write a default constructor and a constructor that takes in parameters to set all the instance variables for the following class. Make sure to set correct values in the default constructor and check for correct values in the parameterized constructor. You may assume mutators have been written and check for correct values.

```java
public class TaterChips
{
    private int numberOfChips;//greater or equal to 0
    private String brandName;//cannot be null
    private double netWT;//greater or equal to 0.0
    //Write your constructors here
    public TaterChips()
    {
        numberOfChips = 30;
        brandName = “Funyuns”;  
        netWT = 26.5;
    }
    public TaterChips(int aNumOfChips, String aBrandName, double aNetWT)
    {
        this.setNumberOfChips(aNumOfChips);
        this.setBrandName(aBrandName);
        this.setNetWT(aNetWT);
    }
}
```

2. Write a static method that takes in two integers and returns the sum. Then implement that in the main method provided.

```java
public class MathSum
{
    //Put your static method here
    public static int Sum(int n1, int n2)
    {
        return n1+n2;
    }
    public static void main(String[] args)
    {
        int number1 = 30;
        int number2 = 12;
        //Implement your method after the equals sign right here
        int number3 = MathSum.Sum(number1,number2);
    }
}
```
3. Create two overloaded methods that set the value of a password instance variable. One method should take in a string and the other should take in an integer. Hint(Integer.parseInt())

public class BriefcaseSecuritySoftware {
    // Assumes passwords can only be numbers
    int password;
    // Put your methods here
    public void setPassword(int aPassword) {
        password = aPassword;
    }
    public void setPassword(String aPassword) {
        password = Integer.parseInt(aPassword);
    }
}
Big Long Question
There will be two questions that will require you to write a class from start to finish. Creating all constructors, accessors, mutators, and methods specified. It will be similar to what you have done in lab.

1. Create a class **Light** that has the following
   a. Two instance variables
      i. isOn – true or false the light is on
      ii. bulbWattage – a non-negative integer value corresponding to the number of watts in the bulb
   b. Two constructors
      i. Default – set the instance variables to a default value
      ii. One that takes in two parameters that will set the instance variables, and check for valid values. The order of these parameters need to follow the order found in part “a”. (IE the “i” should come first, the “ii” should be next, and so on).
   c. Accessors and Mutators for both instance variables
      i. CHECK FOR VALID VALUES

2. Other Methods
   a. toString: This method takes in no parameters and returns a String with the values of the instance variables. It should return a String formatted as:

   “Is On: <<value of isOn>> Bulb Wattage: <<value of bulbWattage>>”

   b. equals: This method takes in an instance of Light and returns true only if all properties are equal.
   c. copyLight: This static method takes in an instance of Light and returns a new instance of Light with the same properties.
   d. turnOnLight: Sets the value isOn to true
   e. turnOffLight: Sets the value of isOn to false

```java
public class Light //define the class
{
    //instance varaibles
    private boolean isOn;
    private int bulbWattage;
    //default constructor
    public Light()
    {
        isOn = false;
        bulbWattage = 50;
    }
    public Light(boolean aIsOn, int aBulbWattage)
    {
        this.setIsOn(aIsOn);
    }
```
this.setBulbWattage(aBulbWattage);

//Accessors
public boolean getIsOn()
{
    return isOn;
}
public int getBulbWattage()
{
    return bulbWattage;
}

//Mutators
public void setIsOn(boolean aIsOn)
{
    isOn = aIsOn;
}
public void setBulbWattage(int aWattage)
{
    if(aWattage >= 0)
        bulbWattage = aWattage;
    else
        bulbWattage = 0;
}

//Methods
toString()
{
    return "Is On: " + isOn + " Bulb Wattage: " + bulbWattage;
}
public boolean equals(Light aLight)
{
    return aLight != null &&
            this.isOn == aLight.getIsOn() &&
            this.bulbWattage == aLight.getBulbWattage();
}
public static Light copyLight(Light aLight)
{
    return new Light(aLight.getIsOn(), aLight.getBulbWattage());
}
public void turnOnLight()
{
    isOn = true;
}
public void turnOffLight()
{
    isOn = false;
}
Objects and Arrays

This chapter focused on arrays and in this exam it will be on arrays of objects. You may expect only 1D arrays and no sorting questions.

1. Assume we have an object named Car which has the attributes make, model, and price of type String, String, and double respectively. The object Car also has accessors and mutators such as getMake, setMake, getPrice, setPrice, etc. Given the following code write a method that returns the instance of a Car with the smallest price from the array of cars. You may not assume every element of the array has been constructed. Also if the array is empty the method should return null.

```java
public class CarLot
{
    private Car[] cars;
    //Put your code here

    public Car getLowestPrice()
    {
        Car ret = null;
        for(int i=0;i<cars.length;i++)
        {
            if(cars[i] != null && ret == null) {
                ret = cars[i];
            } else if(cars[i] != null && cars[i].getPrice() < ret.getPrice()) {
                ret = cars[i];
            }
        }
        return ret;
    }
}
```
Inheritance and Polymorphism.
This chapter was about inheritance, polymorphism, interfaces and abstract classes. I won’t be testing anything on abstract classes or interfaces.

1. We have a class Person that has the instance variable name and age, and has a constructor that takes both of those values in as parameters. Fill in the rest of the class Employee with the proper constructors: A default constructor that sets the employee number to a default value and also calls the Person’s default constructor. Another constructor that takes in the employee number along with the name and age (in that order), and calls the parent’s constructor parameterized.

```java
public class Employee extends Person { //don’t forget to put something here
    int employeeNumber; //Can be any integer
    //Put your constructors here
    public Employee()
    {
        super();
        employeeNumber = -1;
    }
    public Employee(int aNumber, String aName, int anAge)
    {
        super(aName, anAge);
        employeeNumber = aNumber;
    }
}
```

2. We have a class Animal that contains the method public void printInfo(). Fill in the rest of the class Gopher by overriding the parent’s printInfo() method. It should call the Animal’s printInfo() method along with also printing the gopher’s address.

```java
public class Gopher extends Animal { //don’t forget to put something here
    String holeAddress;
    //Put the overridden method here
    public void printInfo()
    {
        super.printInfo();
        System.out.println(“Address: “+holeAddress);
    }
}
```
Exceptions (FINAL EXAM ONLY)
This chapter was on exceptions. You may expect to have questions creating exceptions, writing methods that throw exceptions, or writing code that uses methods that throw exceptions that require a try and catch block.

1. Create an exception class called FatalCheeseException that has two constructors: the default that calls the parent’s constructor with an error message, and another one that takes in a message string that is passed to the parent’s constructor.

public class FatalCheeseException extends Exception //Don’t forget to put stuff here
{
    public FatalCheeseException()
    {
        super("Fatal Cheese Exception! It’s gone bad!");
    }
    public FatalCheeseException(String message)
    {
        super(message);
    }
}
2. Write a method `eatCheese` in the given class `Cheese`. This method returns no values and takes in two parameters corresponding to the current month and year. Furthermore, this method could raise an exception if the current year or month is greater than the expiration date.

```java
public class Cheese {
    private String name;
    private int expMonth;
    private int expYear;

    //Write Method here
    public void eatCheese(int currentMonth, int currentYear) throws FatalCheeseException {
        if (currentYear < expYear) {
            System.out.println("Mmm that's some good cheese");
        } else if (currentYear == expYear && currentMonth <= expMonth) {
            System.out.println("Mmm that's some good cheese");
        } else {
            throw new FatalCheeseException();
        }
    }
}
```
3. We have a class **Cheese** that has three instance variables name, expMonth, expDate. Along with those variables it has all of the accessors and mutators with error checking. This class also has a method **eatCheese** that can throw a **FatalCheeseException**. The method takes in a month and a year that corresponds to the current month and year. In the provided main method create a new instance of cheese setting its name, expMonth, and expYear. Also call its eatCheese method. Make sure if an exception occurs make sure to print the exception’s message.

```java
public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);
    System.out.println("I'm going to eat this cheese, but let me check if it's good. Enter the current month and year");
    int month = keyboard.nextInt();
    int year = keyboard.nextInt();
    //Put your code here calling eatCheese and handling the possible exception
    Cheese brie = new Cheese();
    brie.setExpMonth(12);
    brie.setExpYear(2012);
    try {
        brie.eatCheese(month,year);
    }
    catch(FatalCheeseException e) {
        System.out.println(e.getMessage());
    }
}
```