***CSC 3020***

***Java Programming***

**Assignment 04**

**40 points**

**Due 10/23/2023 (11:45 A.M.)**

Assignment Objectives:

■■ To demonstrate how to define classes and create objects.

■■ To use UML graphical notation to describe classes and objects.

■■ To distinguish between object reference variables and primitive-datatype variables.

■■ To distinguish between instance and static variables and methods.

■■ To define private data fields with appropriate getter and setter methods.

■■ To create immutable objects from immutable classes to protect the contents of objects.

■■ To use the keyword **this** to refer to the calling object itself.

**Solution to this assignment will not be posted on Canvas; however, any question can be discussed in the class upon request of a student.**

All assignments must be submitted by the Canvas. **No email or hard copy** is accepted. You must follow the following format:

1. For non-programming questions, use a word file to type your answers. Don’t use the text box on the Canvas to answer the questions or to write comments, we will not read it.
2. State your answer clearly.
3. For programming questions, include only the source file for each problem.
4. Submit your file to the Canvas. You must submit your assignment on time; otherwise, you will receive zero. In addition, you cannot submit your file more than one time.
5. There will be several folders on the Canvas. You need to upload your file(s) using the correct folder on the Canvas.
6. Name each file: “Assignment Number(Question number(s))”.
7. To upload your file(s):

* In Course Navigation, click the Assignments link.
* Click the title of the assignment.
* Click the **Submit** Assignment button.
* Add **File**. ...
* Add Another **File**. ...
* **Submit** Assignment. ...
* View **Submission**.

**It is your responsibility to make sure that each file is uploaded correctly. If you uploaded a wrong file, you receive zero; files will not be accepted after due date even if you have a prove that the file is created before the due date.**

**Make sure you review the Cheating & Plagiarism policy on Canvas.**

**Answer questions 1 to 3 in a separate single .txt file; include only your answers with questions numbers. Write a program for each of Q.4 - Q.6; save each program in a .txt file. Submit total of 4 .txt files by the due date.**

**Q01. (10 points - 1 point each)**

1. When will a class have a default constructor?

A class has a default constructor when a constructor is not explicitly defined. Java automatically defines a constructor for you if one is not explicitly defined.

1. What is an anonymous object?  
   An anonymous object is an objective that is not assigned a variable when it is created.
2. What is NullPointerException?  
   The NullPointerException is what happens when you try to access an object that does not reference any instance of an object.
3. What is wrong in the following code?

1 **class** Test {

2 **public** **static** **void** main(String[] args) {

3 A a = **new** A(); //the issue is here, the code does not have an input for the constructor, and the constructor has no default value, therefore attempting to call it causes an error.

4 a.print();

5 }

6 }

7

8 **class** A {

9 String s;

10

11 A(String newS) {

12 s = newS;

13 }

14

15 **public** **void** print() {

16 System.out.print(s);

17 }

18 }

1. What is the output of the following code?

**public** **class** A {

**boolean** x;

**public** **static** **void** main(String[] args) {

A a = **new** A();

System.out.println(a.x);

}

}

//the output of this code is ‘false’.

1. a. Can you invoke an instance method or use an instance variable from a static method?   
   Yes (if you have an instance of the class).   
   b. Can you invoke a static method or use a static variable from an instance method?   
   Yes.
2. In the following code, radius is private in the Circle class, and myCircle is an object of the Circle class. Does the code cause any problems? If so, explain why.

**public** **class** Circle {

**private** **double** radius = 1;

/\*\* Find the area of this circle \*/

**public** **double** getArea() {

**return** radius \* radius \* Math.PI;

}

**public** **static** **void** main(String[] args) {

Circle myCircle = **new** Circle();

System.out.println("Radius is " + myCircle.radius);

}

}

//The code outputs ‘Radius is 1.0’

//Remember, radius is inaccessible outside the context of the class, this could cause issues in the future.

1. Describe the difference between passing a parameter of a primitive type and passing a parameter of a reference type. Show the output of the following programs:  
     
   Passing a parameter of a reference type passes the memory address of that reference type. Passing a parameter of a primitive type passes the actual value itself.

**public** **class** Test {

**public** **static** **void** main(String[] args) {

Count myCount = **new** Count();

**int** times = 0;

**for** (**int** i = 0; i < 100; i++)

increment(myCount, times);

System.out.println("count is " + myCount.count);

System.out.println("times is " + times);

}

**public** **static** **void** increment(Count c, **int** times) {

c.count++;

times++;

}

}

**class** Count {

**public** **int** count;

**public** Count(**int** c) {

count = c;

}

**public** Count() {

count = 1;

}

}

//The output of the program is:  
//”Count is 101”

//times is 0

1. Is the following class immutable?

**public** **class** A {

**private** **int**[] values;

**public** **int**[] getValues() {

**return** values;

}

}

No the above class is not immutable, it can be changed after the class is created, this is because the keyword final is not used before the internal values of the class.

1. What is wrong in the following code? Corrected. CORRECT IT YOU MEAN.

1 **public** **class** C {

2 **private** **int** p;

3

4 **public** C() {

6 **this**(0); //this is in the wrong place, put it first.

5 System.out.println("C's no-arg constructor

invoked");

7 }

8

9 **public** C(**int** p) {

10 this.p = p; //gotta change the local instance of p. See my comment below.

11 }

12

13 **public** **void** setP(**int** p) {

14 this.p = p;//without including the this.p it doesn’t change the instance, it changes the entire parameter across all classes. Silly.

15 }

16 }

**Q02. (4 points)**

Suppose that the class F is defined in (a). Let f be an instance of F.

Which of the statements in (b) are correct?

(a)

**public** **class** F {

**int** i;

**static** String s;

**void** imethod() {

}

**static** **void** smethod() {

}

}

(b)

System.out.println(f.i); //incorrect

System.out.println(f.s); //incorrect

f.imethod(); //correct

f.smethod(); //incorrect

System.out.println(F.i); //incorrect

System.out.println(F.s); //correct

F.imethod(); //incorrect

F.smethod(); //correct

//static variables can be accessed through class names OR instances

**Q03. (4 points)**

Show the output of the following code:

(a)

**public** **class** Test {

**public** **static** **void** main(String[] args) {

**int**[] a = {1, 2};

swap(a[0], a[1]);

System.out.println("a[1] = " + a[1]

+ " a[0] = " + a[0]);

}

**public** **static** **void** swap(**int** n1, **int** n2) {

**int** temp = n1;

n1 = n2;

n2 = temp;

}

}

//The output of the above code is:  
//a[1] =2 a[0]=1

(b)

**public** **class** Test {

**public** **static** **void** main(String[] args) {

**int**[] a = {1, 2};

swap(a);

System.out.println("a[1] = " + a[1]

+ " a[0] = " + a[0]);

}

**public** **static** **void** swap(**int**[] a) {

**int** temp = a[0];

a[0] = a[1];

a[1] = temp;

}

}

//the output of the above code is:

//a[1]=1 a[0]=2

(c)

**public** **class** Test {

**public** **static** **void** main(String[] args) {

T t = **new** T();

swap(t);

System.out.println("e1 = " + t.e1

+ " e2 = " + t.e2);

}

**public** **static** **void** swap(T t) {

**int** temp = t.e1;

t.e1 = t.e2;

t.e2 = temp;

}

}

**class** T {

**int** e1 = 1;

**int** e2 = 2;

}

//the output of the above code is:  
//e1 = 2 e2 = 1

(d)

**public** **class** Test {

**public** **static** **void** main(String[] args) {

T t1 = **new** T();

T t2 = **new** T();

System.out.println("t1's i = " +

t1.i + " and j = " + t1.j);

System.out.println("t2's i = " +

t2.i + " and j = " + t2.j);

}

}

**class** T {

**static** **int** i = 1;

**int** j = 1;

T() {

i++;

j = 1;

}

}

//the output of the above code:  
//t1’s i = 3 and j = 1

//t2’s i = 3 and j = 1

**Programming Questions**

For the programming questions, please use the class name given in each question; include the class UML diagram in the word file; include the test class (with main method) in the same file that has the primary class (Time, MyInteger, and Queue).

**Q04. (6 points)**

Design a class named **Time**. The class contains:

* The data fields **hour**, **minute**, and **second** that represent a time.
* A no-arg constructor that creates a **Time** object for the current time. (The values of the data fields will represent the current time.)
* A constructor that constructs a **Time** object with a specified elapsed time since midnight, January 1, 1970, in milliseconds. (The values of the data fields will represent this time.)
* A constructor that constructs a **Time** object with the specified hour, minute, and second.
* Three getter methods for the data fields **hour**, **minute**, and **second**, respectively.
* A method named **setTime(long elapseTime)** that sets a new time for the object using the elapsed time. For example, if the elapsed time is **555550000** milliseconds, the hour is **10**, the minute is **19**, and the second is **10**.

Draw the UML diagram for the class then implement the class. Write a test program that creates three **Time** objects (using **new Time()**, **new** **Time(555550000)**, and **new Time(5, 23, 55)**) and displays their hour, minute, and second in the format hour:minute:second.

(*Hint*: For the no-arg constructor, the current time can be obtained using **System.currentTimeMillis().**

The currentTimeMillis method in the System class returns the current time in milliseconds elapsed since the time midnight, January 1, 1970 GMT (Assume the time is in GMT) .

Here is a sample run:

16:16:27

10:19:10

5:23:55

UML DIAGRAM:

|  |
| --- |
| Time |
| +Hour: int +minute:int  +second:int |
| +Time()  + Time(elapsedMillis:int)  +Time(hour:int, minute:int, second:int)  +gethour():int  +getMinute():int  +getSecond():int  +setTime(elapsedMillis:int):void |

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Design a class named Time. The class contains:

• The data fields hour, minute, and second that represent a time.

• A no-arg constructor that creates a Time object for the current time. (The values of the data fields will represent the current time.)

• A constructor that constructs a Time object with a specified elapsed time since midnight, January 1, 1970, in milliseconds. (The values of the data fields will represent this time.)

• A constructor that constructs a Time object with the specified hour, minute, and second.

• Three getter methods for the data fields hour, minute, and second, respectively.

• A method named setTime(long elapseTime) that sets a new time for the object using the elapsed time. For example, if the elapsed time is 555550000 milliseconds, the hour is 10, the minute is 19, and the second is 10.

Draw the UML diagram for the class then implement the class. Write a test program that creates three Time objects (using new Time(), new Time(555550000), and new Time(5, 23, 55)) and displays their hour, minute, and second in the format hour:minute:second.

(Hint: For the no-arg constructor, the current time can be obtained using System.currentTimeMillis().

The currentTimeMillis method in the System class returns the current time in milliseconds elapsed since the time midnight, January 1, 1970 GMT (Assume the time is in GMT) .

Here is a sample run:

16:16:27

10:19:10

5:23:55

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public class Time {

//Here are the data fields for hour, minute and second

int hour;

int minute;

int second;

//No argument constructor that creates the time obect for the currentTime

public Time() {

long currentTimeMillis = System.currentTimeMillis();

setTime((int)currentTimeMillis);

}

//Constructor to be used for setting a determined time in miliseconds.

//Why? Look, it comes in milliseconds, we're using that

//I am too lazy to convert it every time.

public Time(int elapsedMillis) {

setTime(elapsedMillis);

}

//constructor for inputting in each time manually.

public Time(int hour, int minute, int second) {

this.hour = hour;

this.minute = minute;

this.second = second;

}

//getter method section

public int getHour() {

return hour; //yup that's the hour

}

public int getMinute() {

return minute; //mhm that's the minute

}

public int getSecond() {

return second; //you betcha that's the second

}

//okay, math time.

//This function sets time, it works in conjunction with the

//punlic constructors that create time for the current time.

public void setTime(int elapsedMillis) {

int totalSeconds = elapsedMillis / 1000; //miliseconds to seconds

second = totalSeconds % 60; //seconds

int totalMinutes = totalSeconds / 60;

minute = totalMinutes % 60; //minutes

int totalHours = totalMinutes / 60;

hour = totalHours % 24; //hours

}

//degbugging machine test program

public static void main(String[] args) {

Time currentTime = new Time();

Time time1 = new Time(555550000);

Time time2 = new Time(5, 23, 55);

//print the output to make sure it actually works.

System.out.println(currentTime.getHour() + ":" + currentTime.getMinute() + ":" + currentTime.getSecond());

System.out.println(time1.getHour() + ":" + time1.getMinute() + ":" + time1.getSecond());

System.out.println(time2.getHour() + ":" + time2.getMinute() + ":" + time2.getSecond());

}

}

**Q05. (8 points)**

Design a class named **MyInteger**. The class contains:

* An **int** data field named **value** that stores the **int** value represented by this object.
* A constructor that creates a **MyInteger** object for the specified **int** value.
* A getter method that returns the **int** value.
* The methods **isEven()**, **isOdd()**, and **isPrime()** that return **true** if the value in this object is even, odd, or prime, respectively.
* The static methods **isEven(int)**, **isOdd(int)**, and **isPrime(int)** that return **true** if the specified value is even, odd, or prime, respectively.
* The static methods **isEven(MyInteger)**, **isOdd(MyInteger)**, and **isPrime(MyInteger)** that return **true** if the specified value is even, odd, or prime, respectively.
* The methods **equals(int)** and **equals(MyInteger)** that return **true** if the value in this object is equal to the specified value.
* A static method **parseInt(char[])** that converts an array of numeric characters to an **int** value. Do not use conversion functions to convert the numeric characters to an int.
* A static method **parseInt(String)** that converts a string into an **int** value. Do not use conversion functions to convert the numeric string to an int.

Draw the UML diagram for the class then implement the class. Write a client program that tests all methods in the class.

Here is a sample run for n1 = 5, n2 = 24, numeric string "3539" and numeric character {'3', '5', '3', '9'}:

n1 is even? false

n1 is prime? true

15 is prime? false

3539

3539

n2 is odd? false

45 is odd? true

n1 is equal to n2? false

n1 is equal to 5? True

UML Diagram:

|  |
| --- |
| MyInteger |
| +value:int |
| +MyInteger(value:int)  +getValue():int  +isEven():boolean  +isOdd(): Boolean  +isPrime():Boolean  \_equals(int:Boolean  +equals(MyInteger):Boolean  +static isEven(int);Boolean  +static isOdd(int):Boolean  +static isPrime(int):Boolean  +static isOdd(MyInteger):Boolean  +static isEven(MyInteger):Boolean  +static parseInt(char[]): int  +static parseInt(String):int |

**/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Design a class named MyInteger. The class contains:**

**• An int data field named value that stores the int value represented by this object.**

**• A constructor that creates a MyInteger object for the specified int value.**

**• A getter method that returns the int value.**

**• The methods isEven(), isOdd(), and isPrime() that return true if the value in this object is even, odd, or prime, respectively.**

**• The static methods isEven(int), isOdd(int), and isPrime(int) that return true if the specified value is even, odd, or prime, respectively.**

**• The static methods isEven(MyInteger), isOdd(MyInteger), and isPrime(MyInteger) that return true if the specified value is even, odd, or prime, respectively.**

**• The methods equals(int) and equals(MyInteger) that return true if the value in this object is equal to the specified value.**

**• A static method parseInt(char[]) that converts an array of numeric characters to an int value. Do not use conversion functions to convert the numeric characters to an int.**

**• A static method parseInt(String) that converts a string into an int value. Do not use conversion functions to convert the numeric string to an int.**

**Draw the UML diagram for the class then implement the class. Write a client program that tests all methods in the class.**

**Here is a sample run for n1 = 5, n2 = 24, numeric string "3539" and numeric character {'3', '5', '3', '9'}:**

**n1 is even? false**

**n1 is prime? true**

**15 is prime? false**

**3539**

**3539**

**n2 is odd? false**

**45 is odd? true**

**n1 is equal to n2? false**

**n1 is equal to 5? True**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/**

**//Remember, to rename this, everything in online GDB must be renamed to Main because**

**//that is how it compiles it**

**public class MyInteger**

**{**

**int value; //Create an indt data filed namd value that stores the int**

**//value represented by this object**

**//a constructor that creates a MyInteger object for the specified int value**

**public MyInteger(int specifiedIntValue){**

**this.value = specifiedIntValue;**

**}**

**//• A getter method that returns the int value.**

**public int getValue(){**

**return this.value;**

**}**

**//The methods i• The methods isEven(), isOdd(), and isPrime()**

**//that return true if the value in this object is even, odd, or prime, respectively.**

**//why did you specify this second?**

**public static boolean isEven(int valinput){**

**if((valinput % 2) == 0){**

**return true;**

**}else{**

**return false;**

**}**

**}**

**public boolean isEven(){**

**return isEven(this.value);**

**}**

**//just return the opposite**

**public static boolean isOdd(int valinput){**

**if((valinput % 2) == 1){**

**return true;**

**}else{**

**return false;**

**}**

**}**

**public boolean isOdd(){**

**return isOdd(this.value);**

**}**

**//return is prime, somehow?**

**public static boolean isPrime(int n){**

**// Corner case**

**if (n <= 1)**

**return false;**

**// Check from 2 to n/2**

**//I mean there are some numbers that are technically divisible by three ONLY**

**for (int i = 2; i <= n / 2; i++)**

**if (n % i == 0)**

**return false;**

**return true;**

**}**

**public boolean isPrime(){**

**return isPrime(this.value);**

**}**

**//• The methods equals(int) and equals(MyInteger)**

**//that return true if the value in this object is equal to the specified value.**

**public boolean equals(int valinput){**

**if (this.value == valinput){**

**return true;**

**}else{**

**return false;**

**}**

**}**

**public boolean equals(MyInteger valinput){**

**if (this.value == valinput.getValue()){**

**return true;**

**}else{**

**return false;**

**}**

**}**

**//• A static method parseInt(char[]) that converts an array of numeric characters to an int value.**

**//Do not use conversion functions to convert the numeric characters to an int.**

**public static int parseInt(char[] valinput){**

**int temp = 0;**

**for(int i =(valinput.length -1); i>=0 ; i--){**

**int intvalue = valinput[i] - '0';**

**temp += intvalue \* Math.pow(10, i+1);**

**}**

**return temp;**

**}**

**public static int parseInt(String valinput){**

**int temp = 0;**

**for(int i =(valinput.length() -1); i>=0 ; i--){**

**int intvalue = valinput.charAt(i) - '0';**

**temp += intvalue \* Math.pow(10, i+1);**

**}**

**return temp;**

**}**

**//This is the main function so you can actually test it.**

**public static void main(String[] args) {**

**System.out.println("Hello World");**

**}**

**}**

**Q 06. (8 points)**

Design a class named **Queue** for storing integers. Like a stack, a queue holds elements. In a stack, the elements are retrieved in a last-in first-out fashion. In a queue, the elements are

retrieved in a first-in first-out fashion. The class contains:

* An int[] data field named elements that stores the int values in the queue.
* A data field named size that stores the number of elements in the queue.
* A constructor that creates a Queue object with default capacity 8.
* The method enqueue(int v) that adds v into the queue.
* The method dequeue() that removes and returns the element from the queue.
* The method empty() that returns true if the queue is empty.
* The method getSize() that returns the size of the queue.

Implement the class with the initial array size set to 8. The array size will be doubled once the number of the elements exceeds the size. After an element is removed from the beginning of the array, you need to shift all elements in the array one position to the left.

Write a test program that adds 20 numbers from 1 to 20 into the queue then removes these

numbers and displays them.

Here is a sample run after adding 1 to 20 to the queue and removed them:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

UML Diagram:

|  |
| --- |
| Queue |
| +elements: int[]  +size: int |
| +Queue()  +enqueue(v:int):void  +dequeue():int  +empty():Boolean  +getSize():int |

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Design a class named Queue for storing integers. Like a stack, a queue holds elements. In a stack, the elements are retrieved in a last-in first-out fashion. In a queue, the elements are

retrieved in a first-in first-out fashion. The class contains:

• An int[] data field named elements that stores the int values in the queue.

• A data field named size that stores the number of elements in the queue.

• A constructor that creates a Queue object with default capacity 8.

• The method enqueue(int v) that adds v into the queue.

• The method dequeue() that removes and returns the element from the queue.

• The method empty() that returns true if the queue is empty.

• The method getSize() that returns the size of the queue.

Implement the class with the initial array size set to 8. The array size will be doubled once the number of the elements exceeds the size. After an element is removed from the beginning of the array, you need to shift all elements in the array one position to the left.

Write a test program that adds 20 numbers from 1 to 20 into the queue then removes these

numbers and displays them.

Here is a sample run after adding 1 to 20 to the queue and removed them:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public class Queue

{

int[] elements;//An int[] data field named elements that stores the int values in the queue.

int size; //A data field named size that stores the number of elements in the queue.

//• A constructor that creates a Queue object with default capacity 8.

public Queue(){

this(8);//bada bing, create a constructor that calls the other constructor

}

public Queue(int capacity){

this.size = capacity;

this.elements = new int[capacity];

}

//The method enqueue int v that adds v into the queue;

public void enqueue(int v){

//if the last element is full, time to expand the array;

//in java, empty arraysindecides have the value 0 for numeric types.

if(elements[size-1] == 0){

size \*=2;

int[] elementsTemp = new int[size];

System.arraycopy(elements,0,elementsTemp,0,size);

elements = elementsTemp;

}

elements[size] = v;

}

public int dequeue(){

int temp = this.elements[0];

//shift everything backwards

for(int i=1; i<size;i++){

this.elements[i-1] = this.elements[i];

}

return temp;

}

public boolean empty(){

//my the nature of queues, if the front is empty, the whole

//thing is empty, only need to check the front;

if(elements[0] == 0){

return false;

}else{

return true;

}

}

public int getSize(){

return size;

}

//this is the debugging section in case you need to debug...

public static void main(String[] args) {

System.out.println("Hello World");

}

}