Código Malo en Java:

```java
public class RevealingNamesBadExample {
    List<int[]> theList;

    public List<int[]> getThem() {
        List<int[]> result = new ArrayList<>();
        for(int[] x : theList)
            if(x[0] == 4)
                result.add(x);
        return result;
    }
}
```

Código Bueno en Java:

```java
public class RevealingNameBetterExample {
    List<int[]> gameBoard;
    private final int FLAGGED = 4;
    private final int STATUS_VALUE = 0;

    public List<int[]> getFlaggedCells() {
        List<int[]> flaggedCells = new ArrayList<>();
        for(int[] cell : gameBoard)
            if(cell[STATUS_VALUE] == FLAGGED)
                flaggedCells.add(cell);
        return flaggedCells;
    }
}
```

Código Mejorado en Java:

```java
public class RevealingNamesEvenBetterExample {
    List<Cell> gameBoard;

    public List<Cell> getFlaggedCells() {
        List<Cell> flaggedCells = new ArrayList<>();
        for(Cell cell : gameBoard)
            if(cell.isFlagged())
                flaggedCells.add(cell);
        return flaggedCells;
    }
}
```

Código Malo en Java:

```java
    public void exampleLO(){
        int O = 0;
        int l = 1;
        int a = 1;
        if(O == l)
            a = O;
        else
            l = 1;
    }

    public static void copyChars(char a1[], char a2[]){
        for(int i = 0; i < a1.length; i++)
            a2[i] = a1[i];
    }
}
```

No se entiende el nombre de las variables utilizadas. No se sabe cuál es el arreglo origen/destino.

Código Malo en Java

```java
    public void example(){
        if(url == "www.google.com" && userStatus == 1)
            performSearch();
    }
```

Se debe sustituir la cadena de caracteres y el valor numérico por constantes.

Código Java Bueno:

```
12  public class SearchebleNames {
13      private static final int SEARCHING = 1;
14      private static final String GOOGLE_URL = "www.google.com";
15      String url;
16      int userStatus;
17
18      public void example(){
            if(url == GOOGLE_URL && userStatus == SEARCHING)
20              performSearch();
21      }
```

Nombres pronunciables. A continuación, un código malo:

```
14  public class PronounceableNames {
15      class DtaRcrd102{
16          private Date genymdhms;
17          private Date modymdhms;
18          private final String pszqint = "102";
19      }
20
21      class Customer{
22          private Date generationTimeStamp;
            private Date modificationTimeStamp;
24          private final String CUSTOMER_ID = "102";
25      }
26  }
```

| **Medidas a Tener en Cuenta** |
| --- |
| Evitar código con desinformación.<br>Escribir nombres “buscables”.<br>Escribir nombres pronunciables. |

# Ejercicio 2.1

Se tiene un profesor de matemática que está molesto porque muchos estudiantes están llegando tarde a sus clases. Decide que a partir de la próxima clase va a contar la cantidad de estudiantes que llegan temprano o en tiempo. Si esta cantidad no supera un mínimo que él establecerá entonces la clase será suspendida.

Implementaremos una función que reciba:

a. La cantidad mínima de estudiantes que tiene que llegar puntual (un entero)

b. El tiempo de llegada de cada estudiante (un arreglo de enteros, que significa el número de minutos que el estudiante llega antes o después de la hora de inicio de la clase. Si es positivo significa que llegó tarde. Cero: llegó justo a tiempo y negativo: llegó más temprano.

La función devolverá un String -> "YES” o "NO” se suspenderá la clase.

**El ejercicio consiste en arreglar el nombrado de esta función**.

```java
public static String classCancelled(int m, int[] arr) {
    int c = 0;
    for (int i = 0; i < arr.length; i++) {
        if (arr[i] <= 0) {
            c++;
        }
    }
    return c < m ? "YES" : "NO";
}
```

Vuelva a implementar esta función utilizando el mismo código pero cumpliendo con las siguientes instrucciones.

a) Cambie los nombres de los argumentos de la función a nombres que los describan mejor.

b) Modifique y mejore el nombre de la variable c.

c) Extraiga los literales "YES" y "NO" a dos constantes dentro de la función.

# Ejercicio 2.2

Dado un número de N dígitos se quiere calcular cuántos de sus dígitos son divisores del número. Implementaremos una función que resuelve este problema. La función que se muestra a continuación recibe el número. Primero obtiene una cadena con los dígitos del número. Luego itera por la cadena, verifica si cada dígito de esta es divisor del número recibido y los cuenta. Al final devuelve el número de dígitos divisores.

Ejemplos:

95 tiene como divisor solo el 5 por tanto el resultado es 1.

107 tiene como divisor solo el 1 por tanto el resultado es 1.

112 tiene los tres dígitos como divisores, el resultado es 3.

**El ejercicio consiste en mejorar el nombrado de la función**.

```java
public static int totalDivisorDigits(int n){
    String d = Integer.toString(n);
    int c = 0;
    for(int i = 0; i < d.length(); i++){
        int cd = Integer.parseInt(Character.toString(d.charAt(i)));
        if(cd != 0 && n % cd == 0)
            c++;
    }
    return c;
}
```

Vuelva a implementar esta función utilizando el mismo código pero cumpliendo con las siguientes instrucciones.

a) Mejore el nombre del argumento.

b) Mejore los nombres de la variables d, c y cd.

# Ejercicio 2.3

Al pequeño Boby le encanta el chocolate. Él va a la tienda con $N (N dólares) en su bolsillo. El precio de cualquier chocolate en la tienda es de $C (C dólares). La tienda ofrece un descuento: por cada M envoltorios de chocolate que se entreguen ellos regalan uno (M >= 2). ¿Cuantos chocolates puede comer Boby?

Implementaremos una función que reciba los datos y devuelva la cantidad de chocolates que puede comer. La función cuenta una cantidad inicial como resultado de dividir el total de dinero que tiene Boby por el precio de los chocolates. Al comprar esa cantidad y comérsela le quedan esos envoltorios. Mientras la cantidad de envoltorios sea mayor que m, boby podrá seguir obteniendo chocolates. Entonces se divide la cantidad de envoltorios por m, esto genera nuevos chocolates que Boby se come y por tanto se suman al total. Se obtiene la cantidad de envoltorios restantes que no pudieron ser cambiados y se le suman los nuevos envoltorios obtenidos. Así sucesivamente hasta que ya no queden suficientes envoltorios para obtener chocolates.

Ejemplo:

N = 6, C = 2, M = 2. Da como resultado 5 chocolates. Primero Boby puede comprar 3 chocolates con los 6 dólares, entonces tiene tres envoltorios. Puede usar dos de ellos para obtener un cuarto chocolate. Ahora puede usar el envoltorio que le sobró junto con el envoltorio del nuevo chocolate y solicitar un quinto chocolate. Finalmente le queda un envoltorio que no podrá usar.

**El ejercicio consiste en mejorar el nombrado de la función**.

```
public static int totalChocoletsToEat(int n, int c, int m){
    int ws = n / c;
    int count = ws;
    while(ws >= m){
        int newC = ws / m;
        count += newC;
        ws = ws % m;
        ws += newC;
    }
    return count;
}
```

Vuelva a implementar esta función utilizando el mismo código pero cumpliendo con las siguientes instrucciones.

a) Mejore los nombres de los argumentos n, c y m.

b) Mejore los nombres de las variables ws y newC.

c) Es opcional mejorar el nombre de la variable count.

**Reglas para nombrar:**

1. No utilizar prefijos para instancias y argumentos.
2. Usar una sola palabra por concepto.
3. No usar el mismo nombre para dos propósitos diferentes.
4. Las clases deben tener un sustantivo como nombre. Ej: Customer, Account.
5. Las funciones deben tener formas verbales como nombre. Ej. postPayment, deletePage.
6. Es confuso tener funciones con nombres diferentes, en diferentes clases, con el mismo propósito (Ej: *get*, *fetch*, *retrieve)* o clases con nombres diferentes que son sinónimos o similares (Ej: *Manager, Driver, Controller*).

# Ejercicio 2.4

Dado un arreglo de enteros positivos, se quiere calcular la mayor diferencia absoluta entre dos de ellos cualesquiera.

Ejemplo:

[1, 2, 5] -> La mayor diferencia absoluta es |1-5| = 4.

a) Cree una clase que reciba en el constructor un arreglo y lo almacene en una variable de instancia. Elija nombres correctos para la clase, argumentos del constructor y para la variable de instancia.

b) Implemente una función dentro de la clase que calcule la mayor diferencia absoluta y la devuelva. Elija nombre correcto para la función y para cualquier variable local que utilice.

# Ejercicio 2.5

Habrá una reunión donde participarán todos los trabajadores de una compañía. Se conoce la cantidad de trabajadores de la compañía. De los trabajadores se conoce la edad y el nombre. Antes de la reunión todos los trabajadores se saludan entre sí con un apretón de manos. Cada trabajador saluda a todos los demás una única vez. Se quiere calcular la edad promedio de los trabajadores que asistirán a la reunión y la cantidad de apretones de manos que se van a llevar a cabo en la reunión.

Modele esta situación teniendo en cuenta los siguientes pasos y utilizando los nombres correctos:

a) Cree una clase que representa al trabajador y que sirva únicamente de estructura de datospara almacenar el nombre y la edad.

b) Cree una clase que tenga como responsabilidad el cálculo del promedio de las edades. Que tenga una función que reciba un arreglo o lista con los trabajadores y que calcule y devuelva el promedio de sus edades.

c) Cree una tercera clase para calcular el número de apretones de manos. Esta clase debe contener una función que recibe el total de trabajadores y devuelve la cantidad de apretones de mano.

**HINT**:

Supongamos que son 4 trabajadores. Como cada uno saluda a los otros 3 entonces 4*3 = 12, pero como cada apretón de manos involucra a 2 trabajadores hay que dividir 12/2 = 6. Este sería el total de saludos.

**Reglas para codificar funciones:**

1. Deben implementar una única funcionalidad. Por ejemplo, una función grantAccess() en un módulo de autenticación debe hacer uso de otra función de determine si el usuarios existe, otra que verifique la contraseña y, si todo ok, genere la sesión de usuario correspondiente en la aplicación.

Código malo:

```
public boolean checkPassword(String userName, String password)
{
    for (User user : database.Users())
    {
        if(user.getName().equals(userName))
        {
            String codedPhrase = user.getPhraseEncodedByPassword();
            String phrase = cryptographer.Decrypt(codedPhrase, password);
            if ("Valid Password".equals(phrase))
            {
                Session.Initialize();
                return true;
            }
        }
    }
    return false;
}
```

Código bueno:

```
public void initializeSession(String userName, String password)
        throws Exception {
    if(checkPassword(userName, password))
        Session.Initialize();
    else
        throw new Exception("Bad credentials");
}

public boolean checkPassword(String userName, String password) {
    User user = database.findUserByName(userName);
    if(user != null){
        return validDecodedPassword(user, password);
    }
    return false;
}

private boolean validDecodedPassword(User user, String password) {
    String codedPhrase = user.getPhraseEncodedByPassword();
    String decodedPassword = cryptographer.Decrypt(codedPhrase, password);
    return decodedPassword.equals(VALID_PASSWORD);
}
```

# Ejercicio 3.1

Dado un intervalo [A, B] se quiere calcular el número de cuadrados perfectos dentro del intervalo. Un número X es cuadrado perfecto si existe otro número menor Y que multiplicado por el mismo de X, o sea Y * Y = X. Por ejemplo 1, 4 y 9 son cuadrados perfectos, 15 no lo es. La entrada comienza con un número N que representa la cantidad de casos de prueba. Seguidamente vendrán N líneas con dos números en cada una. Estos son los extremos del intervalo. Para la salida debe imprimir una línea por cada caso de prueba con un número que será la cantidad de cuadrados perfectos dentro del intervalo. Un ejemplo de entrada y salida para este programa es el siguiente:

Entrada:

2

10 20

25 65

Salida

1

4

En el primer intervalo solo el 16 es cuadrado perfecto y en el segundo intervalo 25, 36, 49 y 64 son los cuadrados perfectos.

En el código que se presenta a continuación tenemos una función main que procesa la entrada y la salida del programa. Esta función al mismo tiempo calcula el número de cuadrados perfecto en el intervalo. Esta es una solución válida, pero queremos mejorarla utilizando el principio de que cada función debe hacer una sola cosa.

```
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    int cases = sc.nextInt();
    for(int i = 0; i < cases; i++){
        int leftExtreme = sc.nextInt();
        int rightExtreme = sc.nextInt();
        int perfectSquares = 0;
        for(int base = 0; (base * base) <= rightExtreme; base++){
            int square = base * base;
            if(square >= leftExtreme && square <= rightExtreme)
                perfectSquares++;
        }
        System.out.println(perfectSquares);
    }
}
```

a) Refactorize esta función separándola en dos. Una llamada main que solo procese la entrada y la salida y otra que calcule el número de cuadrados perfectos.

b) Escoja nombres correctos.

# Ejercicio 3.2

Dada una colección de enteros distintos se quiere calcular la mayor y la menor diferencia absoluta entre dos de ellos cualesquiera.

La entrada estará compuesta por dos líneas: un valor entero con la cantidad de números que tendrá la colección y una segunda línea con todos los números separados por un espacio. La colección puede tener números negativos. La salida será una línea con dos valores separados por un espacio. Primero la mayor diferencia absoluta y después la menor.

Ejemplo:

Entrada:

5

10 -5 50 -6 -80

Salida

130 1

La mayor diferencia absoluta es |50 - (-80)| = 130.
La menor diferencia absoluta es |-5 - (-6)| = 1.

a) Escriba una función *main* que procese la entrada. Esta tiene que leer los datos correctamente.

b) Escriba una segunda función para calcular la mayor diferencia absoluta. Esta función debe recibir la colección de enteros y devolver dicha diferencia.

c) Escriba una tercera función para calcular la menor diferencia absoluta. Al igual que la anterior debe recibir la colección y devolver la menor diferencia.

d) Haga que la función *main* se apoye en las otras dos para calcular ambas diferencias y por último que las imprima.

e) Recuerde nombrar correctamente en todos los casos.

# Ejercicio 3.3

Implemente una función que reciba un arreglo de enteros positivos y devuelva la cantidad de elementos del arreglo que son mayores que el promedio de todos los elementos.

Ejemplo: Para el arreglo [2, 3, 10, 1, 9, 5] el resultado es 2, debido a que el promedio es 5 y hay 2 elementos mayores que son en 10 y el 9.

HINT: No es necesario hacer un *main* que procese la entrada e imprima la salida. Solamente la función que se pide. Esta función puede apoyarse en otra(s) para cumplir con las características de las funciones en la programación limpia.

# Ejercicio 3.4

Implemente una función que reciba un número entero y busque el número primo más cercano a él.

a) Utilice al menos una función auxiliar que reciba un número y diga si este es primo o no.

HINT: Un número es primo si sus únicos divisores son 1 y él mismo.

# Ejercicio 3.5

Implemente una función que reciba un número entero y devuelva *true* si él número es perfecto o *false* si no. Un número es perfecto cuando es igual a la suma de todos sus divisores excepto él mismo.

Ejemplo: el número 28 es perfecto (28 = 1 + 2 + 4 +7 + 14).

# Ejercicio 3.6

En el siguiente código se tiene una clase empleado. Esta clase contiene una función *payAmount* que calcula la cantidad que le deben pagar al empleado de acuerdo a su tipo. Este código tiene los problemas vistos en la clase que puede crecer si aumentan los tipos de empleados. Hace muchas cosas, porque calcula el total a pagar a cada empleado y puede provocar que haya que usar esta misma estructura para otras funciones parecidas.

```
class Employee
{
    private EmployeeType type;
    private int monthlySalary;
    private int commission;
    private int bonus;

    public int payAmount() throws Exception
    {
        switch (type)
        {
            case ENGINEER:
                return monthlySalary;
            case SALESMAN:
                return monthlySalary + commission;
            case PROJECT_MANAGER:
                return monthlySalary + bonus;
            default:
                throw new Exception("Incorrect Employee");
        }
    }
}
public enum EmployeeType { ENGINEER, SALESMAN, PROJECT_MANAGER }
```

a) Cree una clase abstracta *Employee* que contenga una propiedad pública *monthlySalary* y una función abstracta *payAmount*().

b) Escriba tres clases *Engineer*, *SalesMan* y *ProjectManager* que hereden de *Employee* e implementen *payAmount*() correspondiendo a lo que está en la función *payAmount*() del código anterior.

c) Escriba una clase: *EmployeeFactory* con una funcion *makeEmployee*(*EmployeeType* type) que reciba un tipo de empleado y cree la instancia de empleado correspondiente.

# EJERCICIO 4.2

(8 ptos) El siguiente fragmento de código corresponde a una función que recibe un intervalo e imprime los números primos dentro del intervalo y luego los números que son divisibles por 3. La función hace más de una cosa, es larga y compleja. Por todo esto su autor decidió comentarla para que se pudiese entender mejor. Usted debe mejorar este código.

a) Divida el código en tantas funciones como crea necesario para que las funciones sean tan sencillas que no requieran ningún comentario.

b) Elimine todos los comentarios.

```
public static void printPrimesAnd3Multiples(int lower, int upper)
      throws Exception{
    // validate params
    if(lower < 2 || upper < 2 || lower > upper){
        throw new Exception("Invalid Params");
    }
    ArrayList<Integer> primes = new ArrayList<>();
    ArrayList<Integer> multiplesOf3 = new ArrayList<>();
    for(Integer i = lower; i <= upper; i++){
        // check if i is prime
        double sqrt = Math.sqrt(i);
        boolean isPrime = true;
        for(int j = 2; j <= sqrt; j++){
            if(i % j == 0){
                isPrime = false;
                break;
            }
        }
        if(isPrime){
           primes.add(i);
        }
        // check if multiple of 3
        String digits = i.toString();
        int sum = 0;
        for(int j = 0; j < digits.length(); j++){
            sum += Integer.parseInt(Character.toString(digits.charAt(j)));
        }
        if(sum % 3 ==0){
            multiplesOf3.add(i);
        }
    }

    // print primes
    System.out.println("Primes");
    for (Integer prime : primes) {
        System.out.println(prime);
    }
    //print multiple of 3
    System.out.println("Multiples of 3");
    for (Integer mult : multiplesOf3) {
        System.out.println(mult);
    }
}
```