Ejercicio U8_B2_E1

```
@FunctionalInterface
interface TransformadorNumerico {

    public int transformar(int x);
}
class App {
    public static void main(String[] args) {
        TransformadorNumerico tn = x -> x * x;
        System.out.println(tn.transformar(4));
    }
}
```

observa que en la expresión lambda al constar de sólo una sentencia en su cuerpo se pudo omitir paréntesis, llaves y return. Expresiones lambda simples como la anterior son muy usadas. Con llaves y return se ven poco.

Ejercicio U8_B2_E2

```
import java.util.function.IntPredicate;

class MiIntPredicate implements IntPredicate{

    @Override
    public boolean test(int i) {
        return i%2==0;
    }

}
public class App{
   public static void main(String[] args){
      int[] lista = {1,2,3,4,5,6,7,8,9};

      System.out.println("Imprimir números int pares:");
      eval(lista, new MiIntPredicate());

    }

   public static void eval(int[] list, IntPredicate predicate) {
      for(int  n: list) {

         if(predicate.test(n)) {
            System.out.println(n + " ");
         }
      }
   }
}
```

reflexión importante: si observamos la solución del ejercicio anterior concluímos que en java una expresión λ crea un objeto que tiene un método test() para lo que hace automáticamente:

1. implementar un interface funcional con la consiguiente sobrescritura del método.
2. Generar una instancia de dicha implementación.

Ejercicio U8_B2_E3

```
import java.util.function.IntPredicate;

public class App{
   public static void main(String[] args){
      int[] lista = {1,2,3,4,5,6,7,8,9};
```

```
      System.out.println("Imprimir números mayores que 5:");
      eval(lista, n-> n>5);

      System.out.println("Imprimir todos los números");
      eval(lista, n-> true);

    }

   public static void eval(int[] list, IntPredicate predicate) {
      for(int  n: list) {

         if(predicate.test(n)) {
            System.out.println(n + " ");
         }
      }
   }
}
```

Ejercicio U8_B2_E4

```
import java.util.function.IntConsumer;

class MiIntConsumer implements IntConsumer{

    @Override
    public void accept(int i) {
       System.out.println(i);
    }

}
public class App{
    public static void main(String[] args) {
        IntConsumer ic = new MiIntConsumer();
        ic.accept(3);

        new MiIntConsumer().accept(4);
    }
}
```

Ejercicio U8_B2_E5

```
import java.util.function.IntSupplier;
class MiIntSupplier implements IntSupplier{
    @Override
    public int getAsInt() {
        return Integer.MAX_VALUE;
    }

}
public class App{
    public static void main(String[] args) {
        IntSupplier i = new MiIntSupplier();
        System.out.println(i.getAsInt());
    }
}
```

Ejercicio U8_B2_E6: hacer una implementación con expresión λ de forma que getAsInt() devuelva un número aleatorio entre 1 y 10

```
import java.util.function.IntSupplier;
public class App{
    public static void main(String[] args) {
        IntSupplier i = ()-> (int) (Math.random() * 10 +1);
        System.out.println(i.getAsInt());
    }
}
```

Ejercicio U8_B2_E7:

```
import java.util.function.IntPredicate;
class MiIntPredicateA implements IntPredicate{
    @Override
    public boolean test(int i) {
        return i<0;
```

```
    }
}

class MiIntPredicateB implements IntPredicate{
    @Override
    public boolean test(int i) {
        return i%2==0;
    }
}
class App{
    public static void main(String[] args) {
        MiIntPredicateA a= new MiIntPredicateA();
        MiIntPredicateB b= new MiIntPredicateB();

        IntPredicate c= a.or(b);//or no devuelve boolean, devuelve un IntPredicate más complejo
        System.out.println(c.test(5));
        //lo mismo pero más compacto y dificil de entender
        System.out.println(a.or(b).test(5));

    }
}
```

Ejercicio U8_B2_E8:

```
import java.util.function.Predicate;

class TodoCierto implements Predicate<Integer>{
    @Override
    public boolean test(Integer i) {
      return true;
    }
}

class ParesCierto implements Predicate<Integer>{
    @Override
    public boolean test(Integer t) {
        return t.intValue()%2==0;
    }
}

class Mayor3Cierto implements Predicate<Integer>{
    @Override
    public boolean test(Integer t) {
        return t.intValue()>3;
    }
}
public class App{
   public static void main(String args[]){
      int[] lista = {1, 2, 3, 4, 5, 6, 7, 8, 9};


      System.out.println("Imprimir todos los números:");
      Predicate<Integer> predicado = new TodoCierto();
      eval(lista, predicado);
```

```
    System.out.println("Imprimir todos  números pares:");
    predicado = new ParesCierto();
    eval(lista, predicado);

    System.out.println("Imprimir números mayores que 3:");
    predicado = new Mayor3Cierto();
    eval(lista, predicado);
  }

  public static void eval(int[] list, Predicate<Integer> predicate) {
    for(int  n: list) {

      if(predicate.test(n)) {
        System.out.println(n + " ");
      }
    }
  }
}
```

Ejercicio U8_B2_E9:

```
import java.util.function.Predicate;


public class App{
  public static void main(String args[]){
    String[] listaPalabras={"hola","adios","zorros","pimiento"};

    System.out.println("Imprimir palabras con más de 5 caracteres:");
    Predicate<String> predicado =s -> s.length()>5;
    eval(listaPalabras, predicado);
   //el método test de Predicate siempre devolverá true

    System.out.println("Imprimir palabras menores que chorizo");

    eval(listaPalabras, s-> s.compareTo("chorizo")<0);


  }

  public static void eval(String[] list, Predicate<String> predicate) {
    for(String  s: list) {

      if(predicate.test(s)) {
        System.out.println(s + " ");
      }
    }
  }
}
```

Ejercicio U8_B2_E10:

Aquí encaja bien el uso de un método genérico ya que no tendría mucho sentido hacer la clase App genérica, por ejemplo, main() no usaría el genérico.

¡OJO CON EL ARRAY DE ENTEROS, por genéricos, el array de int tiene que ser ahora de Integer!

```
import java.util.function.Predicate;
public class App{
```

```
  public static void main(String args[]){
    String[] listaPalabras={"hola","adios","zorros","pimiento"};
    Integer[] listaNumeros={3,4,-5,6,-7};

    System.out.println("Imprimir palabras de más  de 5 car:");
    Predicate<String> predicado = s -> s.length()>5;
    eval(listaPalabras, predicado);

    System.out.println("Imprimir numeros positivos: ");
    eval(listaNumeros, i-> i>0 );

  }

  public static <T> void eval(T[] list, Predicate<T> predicate) {
    for(T e: list) {

      if(predicate.test(e)) {
        System.out.println(e + " ");
      }
    }
  }
}
```

Ejercicio U8_B2_E11:

```
import java.util.function.BiFunction;
class App{
   public static void main(String[] args){
      BiFunction<Double,Integer,Double> xElevadoY = (x,y) -> Math.pow(x,y);
      System.out.println(xElevadoY.apply(2.0,2));
      //System.out.println(xElevadoY.apply(2,2));
      System.out.println(xElevadoY.apply(2.5,2));
   }
}
```