PA1 → due Tuesday

evenOdd

Return an array that contains the exact same numbers as the given array, but rearranged so that all the even numbers come before all the odd numbers. Other than that, the numbers can be in any order. You may modify and return the given array, or make a new array.

```
evenOdd([1, 0, 1, 0, 0, 1, 1]) → [0, 0, 0, 1, 1, 1, 1]
evenOdd([3, 3, 2]) → [2, 3, 3]
evenOdd([2, 2, 2]) → [2, 2, 2]
```

new int[0];

What test cases should we write to confirm that our implementation works?

int[] evenOdd (int[] array) { return null; }

Null
[]
null
null? []?
→ []

1st index 0
last index length-1

[1,2,3,4] → [2,4,1,3], [2,4,3,1],
[4,2,1,3], [4,2,3,1]

Java Inheritance

- Single inheritance of implementation using extends
- Multiple inheritance of interface using implements

class Class1 extends Class2 {
}

What is the annotation @Override used for?

optional → compiler error if params are different

What is the difference between overriding and overloading a method?

overloading → same method name, diff signature (# or type of params)

overriding → over ride (new implementation) of the same method header

Complete the following interface and class definitions for an add method that takes a String as parameter and also an insert method that takes an int and a String as parameters. Leave all method bodies empty.

```
interface StringList {
    void add(String item);
    void insert(int index, String item);
}

class StringListImpl implements StringList {
    @Override
    public void add(String item) { }
    @Override
    public void insert(int index, String item) { }
}
```

assertArrayEquals ([], [])

Arrays.equals ([], [])

↳ if (false) {
    fail ("")
}

↳ Arrays.toString ( )

for ( ) {

↓

2, 4, 1, 3

4, 2, 1, 3

[4, 1, 2, 3]

↑ ↑ ↑

even = true;
false;

fail ( )

13) Given the following definitions:

```
public interface Printable
{
  public abstract String print( boolean duplex );
}
```

```
class Thing1 implements Printable
{
  private String str;

  public Thing1()
  {
    this.str = "Thing 1";
  }

  public String print( boolean duplex )
  {
    return this.str + " duplex = " + duplex;
  }

  public String print()
  {
    // print single sided by default
    return this.print( false );
  }
}
```

overloading

override

```
class Thing2 implements Printable
{
  private String str;

  public Thing2()
  {
    this.str = "Thing 2";
  }

  public String print( boolean duplex )
  {
    return this.str + " duplex = " + duplex;
  }

  public String print( String user )
  {
    System.out.print( user + ": " );

    // print double sided by default
    return this.print( true );
  }
}
```

override

overloading

And the following variable definitions:

```
Thing1 thing1 = new Thing1();
Thing2 thing2 = new Thing2();
Printable printable;
```

① Hint: What does the compiler know about any reference variable at compile time (vs. run time)?

What gets printed with the following statements (each statement is executed in the order it appears). If there is a compile time error, write "Error" and assume that line is commented out when run.

```
System.out.println( thing1.print() );
```
Thing 1 duplex = false

```
System.out.println( thing1.print( "CS11SZZ" ) );
```
Error

```
System.out.println( thing1.print( false ) );
```
Thing1 duplex = false

```
System.out.println( thing2.print() );
```
Error

```
System.out.println( thing2.print( "CS11SZZ" ) );
```
CS11SZZ: Thing 2 duplex = true

```
System.out.println( thing2.print( false ) );
```
Thing 2 duplex = false

② Polymorphism

```
printable = thing1;
System.out.println( printable.print( true) );
```
Thing 1 duplex = true

```
System.out.println( printable.print() );
```
Error

```
System.out.println( printable.print( "CS11SZZ" ) );
```
Error

③ Polymorphism

```
printable = new Thing2();
System.out.println( printable.print( true ) );
```
Thing 2 duplex = true

```
System.out.println( printable.print() );
```
Error

```
System.out.println( printable.print( "CS11SZZ" ) );
```
Error