CSE12 - Lecture 3

Wednesday, October 4, 2023 10:34 AM

Discussion 6pm 6H

Lecture 3

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]
```

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

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

null | null → []

[1, 2, 3, 4] | [2, 4, 1, 3] [2, 4, 3, 1] [4, 2, 1, 3] [4, 2, 3, 1]

assertArrayEquals([], []);

Arrays.equals([], []) → true/false

if (false) {

fail(" " + Arrays.toString(array));

}

Java Inheritance

class Class1 extends Class2 {

}

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

What is the annotation @Override used for?

optional → compile error if parameters are different

What is the difference between overriding and overloading a method?

Override → replaces implementation of a method (same header) in a class

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 {
public abstract void add(String s);
public abstract void insert(int location, String s);
}

class StringListImpl implements StringList {
    @Override
    void add(String s) { }

    @Override
    void insert(int loc, String s) { }
}
```

// overload

void add(int i) { }

Name: _____ PID: _____ Code: 9122

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 );
  }
}
```

override

overload

```
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

overload

And the following variable definitions:

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

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 ...
`System.out.println( thing1.print( "CS11SZZ" ) );`	Error
`System.out.println( thing1.print( false ) );`	Thing 1 ...
`System.out.println( thing2.print() );`	Error
`System.out.println( thing2.print( "CS11SZZ" ) );`	CS11SZZ: Thing 2 ...
`System.out.println( thing2.print( false ) );`	Thing 2 ...
`printable = thing1;`	
`System.out.println( printable.print( true) );`	Thing 1 ...
`System.out.println( printable.print() );`	Error
`System.out.println( printable.print( "CS11SZZ" ) );`	Error
`printable = new Thing2();`	
`System.out.println( printable.print( true ) );`	Thing 2 ...
`System.out.println( printable.print() );`	Error
`System.out.println( printable.print( "CS11SZZ" ) );`	Error