CSE12 - Lecture 3

Friday, April 7, 2023   8:00 AM

int [] a = {1, 2, 3}
int [] b = {1, 2, 3}

a. equals (b) → false

Arrays. equals (a, b)

→ assertArrayEquals ([...], [...]) → true / false

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]

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

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

edge cases → [] , null → []

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

if (   ) {
    fail (" .... ");
} else if (   ) {

}
⋮

Java Inheritance

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

extends Object

Class Class1 extends Class2 {
// all public methods, all fields

What is the annotation @Override used for?

optional → compiler 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

overload → same method name, different parameters → different signature

, or type

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.

default

interface StringList {
    public abstract void add(String s);
    public abstract void insert(int index, String s);
}

→ StringList sl = new StringListImpl();

sl. add (   )

sl. insert (   )

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

    public void insert (int index, String s) { ... }
}

Name: _____   PID: _____   Code: 1568

13) Given the following definitions:

```
public interface Printable
{
  public abstract String print( boolean duplex );
}
```
(handwritten: ↓ arrow pointing to print)

```
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 );
  }
}
```
(handwritten left margin: 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 );
  }
}
```
(handwritten right margin: 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() );
```
(handwritten: Thing 1 duplex false)

```
System.out.println( thing1.print( "CS11SZZ" ) );
```
(handwritten: error)

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

```
System.out.println( thing2.print() );
```
(handwritten: error)

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

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

```
printable = thing1;
```

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

```
System.out.println( printable.print() );
```
(handwritten: error)

```
System.out.println( printable.print( "CS11SZZ" ) );
```
(handwritten: error)

```
printable = new Thing2();
```

```
System.out.println( printable.print( true ) );
```
(handwritten: Thing 2 duplex = true)

```
System.out.println( printable.print() );
```
(handwritten: error)

```
System.out.println( printable.print( "CS11SZZ" ) );
```
(handwritten: error)

6