CS61B Spring 2016 Secret Section 1 Worksheet

CS61B Tutors

Week 1

1 Access Control

The goal is to learn how access control works in Java, across classes, files, and packages. Use the following table as a quick reference. Remember that if an access modifier is omitted, it is **default** by default.

	Same Class	Same Package	Subclass (different package)	World
private	**Yes**	No	No	No
default	**Yes**	**Yes**	No	No
protected	**Yes**	**Yes**	**Yes**	No
public	**Yes**	**Yes**	**Yes**	**Yes**

```
1  package com.cs61b.animal;
2
3  public class Dog {
4      private String name = "Wolf";
5      int age = 35;
6      protected double weight = 123.4;
7      public String nickname = "DumDum";
8
9      public void bark() {
10         System.out.println(name + " barks, Woof!");
11     }
12
13     void sleep() {
14         System.out.println("zzz");
15     }
16
17     protected String getName() {
18         return name;
19     }
20
21     private void changeNickname(String n) {
22         nickname = n;
23     }
24 }
```

```
1  package com.cs61b.animal;
2
3  public class Cat {
4      private Dog d = new Dog();
5
6      public void olderDog() {
7          d.age += 5;
8      }
9
10     public void heavierDog() {
```

```
        d.weight += 10;
    }

    public void changeDogNickname(String newNickname) {
        d.changeNickname(newNickname);
    }

}
```

```
package com.cs61b.robot;

import com.cs61b.animal.Dog;

public class RobotDog extends Dog {

    public void sleepMode() {
        sleep();
    }

    public void sayName1() {
        System.out.println("My name is " + getName());
    }

    public void sayName2() {
        System.out.println("My name is " + name);
    }

    public Cat getCat() {
        return new RobotCat();
    }
}
```

```
package com.cs61b.robot;

import com.cs61b.animal.Cat;

class RobotCat extends Cat {
    public void meow() {
        System.out.println("MEOW!");
    }
}
```

```
package com.cs61b;

public class Main {

    public static void main(String[] args) {
        Dog dog = new Dog();
        Cat cat = new Cat();
        RobotDog robotDog = new RobotDog();
        RobotCat robotCat = new RobotCat();
        Cat catFromDog = robotDog.getCat();
        catFromDog.meow();

        dog.name = "Fluffy";
        dog.age = 125;
        dog.weight = 0.23;
        dog.nickname = "Hamburger";
```

```
17
18          dog.bark();
19          dog.sleep();
20          System.out.println(dog.getName());
21          dog.changeNickname("Burrito");
22
23          cat.olderDog();
24          cat.heavierDog();
25          cat.changeDogNickname("Hamstring");
26
27          robotDog.sleepMode();
28          robotDog.sayName1();
29          robotDog.sayName2();
30      }
31
32  }
```

Your Job: Cross out all lines in above code (Dog, Cat, RobotDog, RobotCat, and Main) that are ILLEGAL, and be able to give brief justifications why.

2 Iterator

```
1   public interface Iterator<T> {
2       boolean hasNext();
3       T next();
4   }
5
6   public interface Iterable<T> {
7       Iterator<T> iterator();
8   }
9
10  public class Dog {
11      private String name = ...;
12      public void bark() {
13          System.out.print("Woof " + name + "!");
14      }
15  }
```

Question 1. Implement the following method, `barkCombinations`, which prints ALL bark combinations of dogs from two dog sets in form of "Woof DOG1! Woof DOG2!". The dogs in dogs1 should bark first. For 3 dogs in dogs1 and 5 dogs in dogs2, there must be 15 bark combinations printed. (Hint: `Set<T>` implements `Iterable<T>`)

```
1   public static void barkCombinations(Set<Dog> dogs1, Set<Dog> dogs2) {
2       // Your code here.
3
4
5
6
7
8
9
10
11
12
13
14
15  }
```

Question 2. Implement the following method `printAllTwice` which prints each element returned by given `String` iterator TWICE. To see how it should work, refer to the `main` method posted below.

```
public static void printAll(Iterator<String> iterator) {
    // Your code here.









}


public static void main(String[] args) {
    List<String> dogs = new ArrayList<>();
    dogs.add("Helium");
    dogs.add("Silver");
    dogs.add("Neumann");
    printAll(dogs.iterator());
}
```

`main` should print:

```
Helium
Helium
Silver
Silver
Neumann
Neumann
```

3 Exceptions

```
public class IntList {
    private int head;
    private IntList tail;

    /* Returns the index of an element in the list */
    public int getIndex(int item){
        int index = 0;
        IntList temp = this;
        while(temp.head != item){
            temp = temp.tail;
            index++;
        }
        return index;
    }
}
```

Question 1. What happens when you call `getIndex(int item)` on an element that is not in the list?

Question 2. Write `getIndexThrowException`, which attempts to get the index of an item, but throws an `IllegalArgumentException` with a useful message if no such item exists in the list. Do *NOT* use `if` statements, `while` loops, `for` loops, or recursion. (Hint: you can use `get(int item)`)

Question 3. Write `getIndexDefaultNegative`, which attempts to get the index of an item, but returns `-1` if no such item exists in the list. Again, do not use `if` statements, `while` loops, `for` loops, or recursion.

```
public class IntList {
    private int head;
    private IntList tail;

    /* Returns the index of an element in the list */
    public int getIndex(int item){
        int index = 0;
        IntList temp = this;
        while(temp.head != item){
            temp = temp.tail;
            index++;
        }
        return index;
    }

    public int getIndexThrowException(int item) {
        // Your code here.









    }

    public int getIndexDefaultNegative(int item) {
        // Your code here.









    }
}
```

4 More Exceptions

```
public class FactorialException extends RuntimeException {
    ...
}

public class WrongNumberException extends Exception {
    ...
}
```

Question 1. Is `FactorialException` a checked exception? What about `WrongNumberException`? Why? Why not?

```
public static int factorial(int n)                          {
    if (n < 0) {
        throw new WrongNumberException();
    }
    if (n <= 1) {
        return 1;
    } else {
        return factorial(n - 1);
    }
}

public static int weirdFactorial(int n) {
    if (n == 50) {
        throw new FactorialException();
    }
    if (n <= 1) {
        return 1;
    } else {
        return weirdFactorial(n - 1);
    }
}
```

Question 2. Is `factorial`'s definition legal? Is there something to be changed? If there is any, fix it!

Question 3. Repeat Question 2 for `weirdFactorial`.

Question 4. Refering to the definition on the previous page for `factorial` and `weirdFactorial`, answer what is printed for each blocks of code below. If the program terminates with an exception, specify which exception occurred. If illegal (doesn't compile), explain why. For numeric results, feel free to just put down "number" instead of performing tedious calculations by hand.

```
//(a)
int answer = factorial(50);
System.out.println(answer);
// ANSWER: _____________________________

//(b)
try {
    int answer = factorial(50);
    System.out.println(answer);
} catch (WrongNumberException ex) {
    System.out.println("Wrong Number!");
}
// ANSWER: _____________________________


//(c)
int answer = weirdFactorial(30);
System.out.println(answer);
// ANSWER: _____________________________


//(d)
int answer = weirdFactorial(60);
System.out.println(answer);
// ANSWER: _____________________________


//(e)
try {
    int answer = weirdFactorial(50);
    System.out.println(answer);
} catch (FactorialException ex) {
    System.out.println("Factorial Exception!");
} catch (Exception ex) {
    System.out.println("Some Exception!");
}
// ANSWER: _____________________________


//(f)
try {
    int answer = weirdFactorial(50);
    System.out.println(answer);
} catch (Exception ex) {
    System.out.println("Some Exception!");
} catch (FactorialException ex) {
    System.out.println("Factorial Exception!");
}
// ANSWER: _____________________________
```