Homework 13

```
static final NaturalNumber NINE = new NaturalNumber2(9);

/**
 * Returns the product of the digits of {@code n}.
 *
 * @param n
 *          {@code NaturalNumber} whose digits to multiply
 * @return the product of the digits of {@code n}
 * @clears n
 * @ensures productOfDigits1 = [product of the digits of n]
 */
private static NaturalNumber productOfDigits1(NaturalNumber n) {

   int remainder, productInt = 1;
   NaturalNumber product = new NaturalNumber2(1);
   NaturalNumber n2 = new NaturalNumber2(n);

   if (n.compareTo(NINE) == 1) {
      remainder = n2.divideBy10();
      productInt = productInt * remainder;
      product = new NaturalNumber2(productInt);
      product.multiply(productOfDigits1(n2));
   } else {
      remainder = n.divideBy10();
      productInt = productInt * remainder;
      product = new NaturalNumber2(productInt);
   }

   return product;

}

/**
 * Returns the product of the digits of {@code n}.
 *
 * @param n
 *          {@code NaturalNumber} whose digits to multiply
 * @return the product of the digits of {@code n}
 * @ensures productOfDigits2 = [product of the digits of n]
 */
private static NaturalNumber productOfDigits2(NaturalNumber n) {

        int remainder, productInt = 1;
```

```
    NaturalNumber product = new NaturalNumber2(1);

    if (n.compareTo(NINE) == 1) {
        remainder = n.divideBy10();
        productInt = productInt * remainder;
        product = new NaturalNumber2(productInt);
        product.multiply(productOfDigits1(n));
    } else {
        remainder = n.divideBy10();
        productInt = productInt * remainder;
        product = new NaturalNumber2(productInt);
    }

    return product;

}

/**
 * Reports the value of {@code n} as an {@code int}, when {@code n} is small
 * enough.
 *
 * @param n
 *          the given {@code NaturalNumber}
 * @return the value
 * @requires n <= Integer.MAX_VALUE
 * @ensures toInt = n
 */
private static int toInt(NaturalNumber n) {

    if (n.canConvertToInt()) {
        return n.toInt();
    } else {
        return -1;
    }
}

/**
 * Reports whether the given tag appears in the given {@code XMLTree}.
 *
 * @param xml
 *          the {@code XMLTree}
 * @param tag
 *          the tag name
 * @return true if the given tag appears in the given {@code XMLTree}, false
 *       otherwise
 * @ensures <pre>
 * findTag =
 *    [true if the given tag appears in the given {@code XMLTree}, false otherwise]
 * </pre>
 */
private static boolean findTag(XMLTree xml, String tag) {
    boolean found = false;

    // the most inefficient loop, but i'm tired
    if (xml.numberOfChildren() > 0) {
        for (int i = 0; i < xml.numberOfChildren(); i++) {
            if (xml.child(i).label().equals(tag)) {
                found = true;
            } else if (!found && xml.child(i).numberOfChildren() > 0) {
                found = findTag(xml.child(i), tag);
            }
        }

    }

    return found;
}
```

i. design-by-contract
    i. Programming around a set of guidelines in order to achieve a desired pre and post condition
ii. Precondition
    i. What is required to be passed to the method before it runs
iii. Postcondition
    i. What the method is required to return
iv. Testing
    i. Making sure the program works the way it should
v. Debugging
    i. Moving line by line through a program, with the intention of finding issues in the code to fix
vi. parameter mode
    i. A tag that describes what the method will do to that argument
vii. Clears
    i. Removing a value from a variable
viii. Replaces
    i. Changing a variable's value with another
ix. Restores
    i. Variable is unchanged
x. Updates
    i. Changing a variable's value
xi. immutable type
    i. Type that is unchangable
xii. primitive type

i. Built in java data types (int, char, double, etc.)

xiii. reference type

i. Any non-primitive data type

xiv. Object

i. An instance of a class

xv. Aliasing

i. When two variables reference the same object

xvi. declared type/static type

i. When a type is cast to a variable and is unchangeable

xvii. object type/dynamic type

i. When a variable type is changeable based on what the variable is being used in

xviii. Implements

i. A keyword used to implement an interface

xix. Extends

i. Showing that a class is inherited from another class

xx. method overriding

i. When a subclass has the same method as a parent class

xxi. subinterface/derived interface/child interface

i. An interface that can extend another interface

xxii. superclass/base class/parent class

i. A class that can be extended from

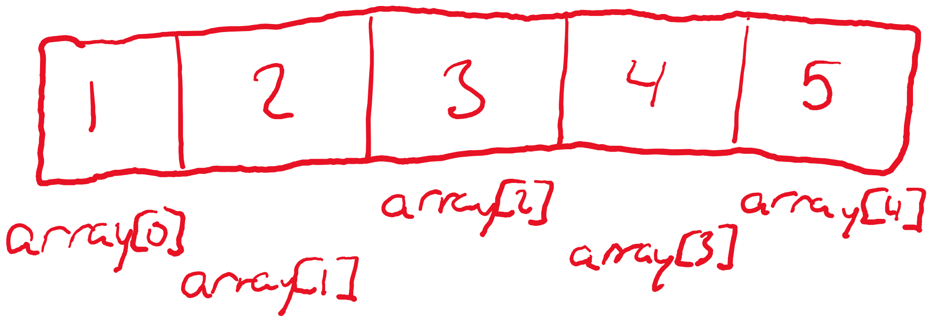
xxiii. Polymorphism

i. When a thing can behave in different ways based on the scenario

xxiv. recursion

i. Calling a method within itself

5.



Line 6 - add parenthesis to array.length to make it array.length()

6. Since NaturalNumber extends NaturalNumber-Kernel, and NaturalNumber-Kernel extends Standard, then that means NaturalNumber also extends Standard.

7. Since C4 extends C3, and C3 implements I2, then C4 also implements I2.

This also means that C3 implements I1 since C3 implements I2, and I2 extends I1.