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;

}

1. design-by-contract
   1. Programming around a set of guidelines in order to achieve a desired pre and post condition
2. Precondition
   1. What is required to be passed to the method before it runs
3. Postcondition
   1. What the method is required to return
4. Testing
   1. Making sure the program works the way it should
5. Debugging
   1. Moving line by line through a program, with the intention of finding issues in the code to fix
6. parameter mode
   1. A tag that describes what the method will do to that argument
7. Clears
   1. Removing a value from a variable
8. Replaces
   1. Changing a variable’s value with another
9. Restores
   1. Variable is unchanged
10. Updates
    1. Changing a variable’s value
11. immutable type
    1. Type that is unchangable
12. primitive type
    1. Built in java data types (int, char, double, etc.)
13. reference type
    1. Any non-primitive data type
14. Object
    1. An instance of a class
15. Aliasing
    1. When two variables reference the same object
16. declared type/static type
    1. When a type is cast to a variable and is unchangeable
17. object type/dynamic type
    1. When a variable type is changeable based on what the variable is being used in
18. Implements
    1. A keyword used to implement an interface
19. Extends
    1. Showing that a class is inherited from another class
20. method overriding
    1. When a subclass has the same method as a parent class
21. subinterface/derived interface/child interface
    1. An interface that can extend another interface
22. superclass/base class/parent class
    1. A class that can be extended from
23. Polymorphism
    1. When a thing can behave in different ways based on the scenario
24. recursion
    1. Calling a method within itself