

91

CSE 2221 — Midterm Exam #1 Spring 2019

This is a closed-book, closed-notes, closed-electronic-device, closed-neighbor exam.

In accordance with The Ohio State University Code of Student Conduct, I certify that I have neither received nor given aid on this examination, that I shall not discuss the contents of this examination with anyone in CSE 2221 who has not already taken the exam, and that I have not recorded and taken from the room any questions or answers from this exam.

Name _____ Signature _____

There are 100 points on the exam. Please write your answers on the test sheets, and of course don't forget to write **and sign** your name on the top sheet. Consider the space allotted as an indication of the expected length of the answer.

1. (20 points, 4 points each) Multiple choice. Circle the best response to each.

1.1. In design-by-contract, the code responsible for making sure that the precondition (requires clause) is true when a method is called is:

- A. the code that implements the method
- B. the code that compiled the method
- ☒ C. the code that calls the method
- D. the code that designed the method

1.2. Consider the following code fragment:

```
int x = 8;  
while ((x / 3) != 2) {  
    x = x - 1;  
}
```

How many times will the body of the loop execute?

- ☒ A. 0
- B. 1
- C. 2
- ☒ D. The loop will never terminate.

1.3. Consider the following method header:

```
private static void triple(int num)
```

and the client code:

```
int number = 11;  
triple(number);
```

The value of variable `number` after the method call:

- ☒ A. is guaranteed to be 11
- B. is guaranteed to be 33
- C. could be either 11 or 33, but nothing else
- D. could be anything; there is not enough information to know

1.4. Consider the following statement:

```
if ((in.nextInt() % 7) == 4) {  
    return false;  
} else {  
    return true;  
}
```

Which of the following statements is it equivalent to?

- A. `return ((in.nextInt() % 7) == 4);`
- ☒ B. `return ((in.nextInt() % 7) != 4);`
- C. `return (((in.nextInt() % 7) != 4) == false);`
- D. None of the above.

1.5. If `x` is an `int` variable, when does the `boolean` expression below evaluate to `true`?

```
((x % 5 != 0) && (x % 2 != 0))
```

- A. When `x` is divisible by 5 or by 2 but not by both.
- B. When `x` is divisible by 10.
- C. When `x` is not divisible by 10.
- ☒ D. When `x` is neither divisible by 5 nor by 2.

2. (24 points) Short Answers

- 2.1. (5 points) Draw a **box** around each of the five **statements** and underline each of the five **expressions** in the following code fragment. Make sure it is clear what is enclosed in boxes and what is underlined.

swap

```

int count = in.nextInteger();

if (count > 0) {
    boolean done = rnd.nextInt() < count;

    while ((count > 0) && !done) {
        count = count - 1;
    }
}
    
```

- 2.2. (8 points) For each of the following statements about XMLTrees, indicate whether it is True or False by circling the appropriate value:

- a. Every node in an XMLTree has a label
 b. Every node in an XMLTree is either a tag or text node
 c. Tag nodes cannot be leaf nodes
 d. A text node can have zero or more attributes

☒ T F
☒ T F
☐ T ☒ F
☐ T ☒ F

- 2.3. (5 points) Consider the following method contract:

```

/**
 * @requires x < 2 * y - 1
 * @ensures  mystery = (y - 1) / (x - 2)
 */
private static int mystery(int x, int y)
    
```

In one short sentence explain what the result of the following call will be and why:

```
int z = mystery(5, 3);
```

the precondition is not satisfied because $x < 2 * y - 1$
 so the method "mystery" will likely get stuck in a loop or possibly even crash. $5 < 2 * 3 - 1$
 $5 < 5$ — not true

2.4. (6 points) Consider the following method implementation and method call:

```
public static int mystery(int i, int j) {  
    i = j;  
    j = j + 1;  
    return i + j;  
}  
...  
int num1 = 3, num2 = 4;  
int result = mystery(num1, num2);
```

After the above call to mystery, what values will the following variables have?

```
num1 = 3  
num2 = 4  
result = 9
```

3. (13 points) Complete the body of hasDuplicates so it satisfies the contract. For full credit, you must use only one **return** statement and no **break** statements.

```
/**  
 * Returns whether the given array has duplicate entries.  
 *  
 * @ensures hasDuplicates =  
 *           [true if at least two entries in numbers are equal,  
 *            false otherwise]  
 */  
private static boolean hasDuplicates(int[] numbers) {  
    boolean atLeastOne = false;  
    int oneEqual = 0;  
    int count = 0;  
    while (count < numbers.length) {  
        int check = count + 1;  
        while (check < numbers.length) {  
            if (numbers[count] == numbers[check]) {  
                oneEqual++;  
            }  
            check++;  
        }  
        count++;  
    }  
    if (oneEqual > 0) {  
        atLeastOne = true;  
    }  
    return atLeastOne;  
}
```

Handwritten notes: *numbers = {0, 1, 2, 0}*, *why count should exit*

4. (14 points) Complete the body of the method `cubeRoot` that computes the cube root ($\sqrt[3]{x}$ or $x^{1/3}$) of the given x within a relative error of no more than the given `epsilon` using Newton iteration (similar to what you did in a recent project). If r is the current guess, the next guess should be $\frac{1}{3}\left(2r + \frac{x}{r^2}\right)$. A good initial guess for $x^{1/3}$ is simply $r = x$, and to ensure that the relative error of the guess r is less than ϵ , your algorithm should keep updating r until $\frac{|r^3 - x|}{x} < \epsilon^3$.

For full credit, you must use a **while** loop (with no **break** statements) in your solution (no **for** or **do-while** loops), and you must use only one **return** statement. You may use the `Math.abs()` static method, but do **not** use the `Math.pow()` function. Make sure to write Java expressions in your code (don't just copy the math expressions from the question).

```
/**
 * Computes and returns an estimate of the cube root of x to
 * within relative error epsilon.
 *
 * @requires x >= 0 and epsilon > 0
 * @ensures [cubeRoot is within relative error epsilon of the
 *          actual cube root of x]
 */
```

```
private static double cubeRoot(double x, double epsilon) {
```

```
    double r = x;
```

```
    while ((Math.abs((r*r*r) - x)/x) > (epsilon * epsilon * epsilon)) {
```

```
        r = ((10/3) * (2*r + (x/(r*r))))/3;
```

```
    }
```

```
    return r;
```

```
}
```

5. (13 points) Complete the body of the method `replaceAll` so it satisfies the contract. Use the `String` instance methods `length()` to find the length of a string and `charAt(int)` to get the character at a given position in the string. You cannot use any of the `String` class *replace* methods. For full credit, you must use only one **return** statement and no **break** statements.

```
/**
 * Returns a new String where all occurrences of oldCh in str
 * have been replaced with newCh.
 *
 * @ensures replaceAll =
 *           [str with all occurrences of oldCh replaced by newCh]
 */
```

```
private static String replaceAll(
    String str, char oldCh, char newCh) {
```

```
    int count = 0;
```

```
    while (count < str.length()) {
```

```
        if (str.charAt(count) == oldCh) {
```

```
            str.charAt(count) = newCh;
```

```
        }
```

```
        count++;
```

```
    }
```

```
    return str;
```

6. (16 points) Tracing

- 6.1. (12 points) On the next page, complete the tracing table as you learned in this class. Make sure you trace *every* variable (except for `s`, which does not change) through *every* statement that is executed and indicate any iterations or cases where code is skipped by drawing a line or leaving a blank.

Code	State (Variable Values)
<pre> /** * Reports whether the given string * contains at least one space. */ private static boolean hasSpace(String s) { int pos = 0; boolean hasSpace = false; </pre>	
	<pre> s = "X Y" pos = 0 hasSpace = false </pre>
<pre>while (pos < s.length()) {</pre>	
	<pre> pos = 0 1 2 hasSpace = false false true </pre>
<pre> if (s.charAt(pos) == ' ') {</pre>	
	<pre> pos = 1 hasSpace = false </pre>
<pre> hasSpace = true;</pre>	
	<pre> pos = 1 hasSpace = true </pre>
<pre> } else {</pre>	
	<pre> pos = 0 2 hasSpace = false true </pre>
<pre> hasSpace = false;</pre>	
	<pre> pos = 0 2 hasSpace = false false </pre>
<pre> }</pre>	
	<pre> pos = 0 1 2 hasSpace = false true false </pre>
<pre> pos = pos + 1;</pre>	
	<pre> pos = 1 2 3 hasSpace = false true false </pre>
<pre>}</pre>	
	<pre> pos = 3 hasSpace = 4 false </pre>
<pre>return hasSpace;</pre>	

6.2. (4 pts) Is the code above a correct implementation of the informal contract for `hasSpace`? Briefly justify your answer.

✓ No, it's not. It does set `hasSpace` to true, but then it goes through the loop again and since the last position in page 7 the string was not a space, it sets `hasSpace` to false.

This page may be used as scratch space.

86

CSE 2221 — Midterm Exam #2 Spring 2019

This is a closed-book, closed-notes, closed-electronic-device, closed-neighbor exam.

In accordance with The Ohio State University Code of Student Conduct, I certify that I have neither received nor given aid on this examination, that I shall not discuss the contents of this examination with anyone in CSE 2221 who has not already taken the exam, and that I have not recorded and taken from the room any questions or answers from this exam.

Name 112 11

Signature _____

There are 100 points on the exam. Please write your answers on the test sheets, and of course don't forget to write **and sign** your name on the top sheet. Consider the space allotted as an indication of the expected length of the answer. Summaries of relevant Javadoc APIs are included at the end of the exam. If you need to look up the Javadoc API for anything else, your instructor will have a browser open on a computer in the classroom for you to use. Please do not detach any pages (including the last one) from the exam.

1. (10 points; 1 point for each letter) Write the letter of each term in front of the description that best matches it. Each letter should be used exactly once.

~~A.~~ Aliases

~~F.~~ Debugging

B. Declared type

~~G.~~ Extends

~~C.~~ Implements

H. Method overloading

~~D.~~ Method overriding

I. Object type

~~E.~~ Reference type

~~J.~~ Testing

☒ Relation between a class and an interface in Java

☒ Showing that code is defective by executing it

☒ Finding and repairing a defect in code ☒ The name of the inheritance relation in Java

☒ Providing two or more methods with the same name but different formal parameters ☒ The type of a variable used by the Java compiler to determine how the variable can be used

☒ Type of variable representing a memory address ☒ Multiple references to the same object

☒ The type of a variable used by Java to decide which method body to use in the presence of overriding ☒ Providing a new implementation for an inherited method

2. (32 points) Short Answers

2.1. (4 points; 2 points each) Consider the code segment below:

```
NaturalNumber n1 = new NaturalNumber1L();
NaturalNumber n2 = new NaturalNumber2(n1);
```

a. What is the *declared* (or *static*) type of n2?

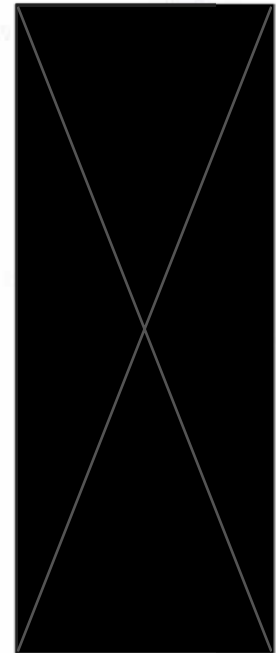


b. What is the *object* (or *dynamic*) type of n2?



2.2. (8 points; 1 point each) For each of the following statements indicate whether it is True or False by circling the appropriate value.

- a. Instance methods can only be private
- b. Instance methods may be static or non-static
- c. Instance methods may have any return type, including **void**
- d. In the call `n.foo(x)`, `x` is known as the receiver of the call
- e. The reference value of a reference variable is related to the object's location in memory
- f. The reference value of a reference variable is related to the object's mathematical value
- g. If you have a thorough test plan and your method passes all the tests in it, you have proven that your method is correct
- h. Testing and debugging are synonyms for the same process



2.3. (8 points; 2 points each) Consider the following method:

```
static void foo(int i, String s, NaturalNumber n, int[] a) {  
    i += 3;  
    s = "goodbye";  
    n.increment();  
    n = new NaturalNumber2(14);  
    a[0] = a.length;  
}
```

and the client code below:

```
int x = 10;  
String str = "hello";  
NaturalNumber num = new NaturalNumber2(7);  
int[] array = { 1, 2, 3, 4 };  
foo(x, str, num, array);
```

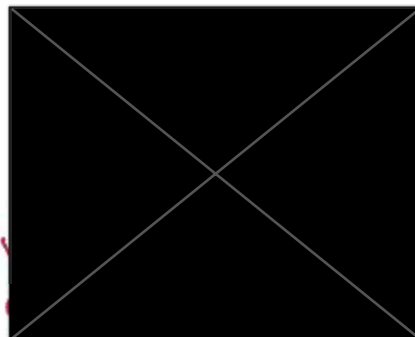
What are the values of variables `x`, `str`, `num`, and `array` after the call to method `foo`?

`x` =

`str` =

`num` =

`array` =



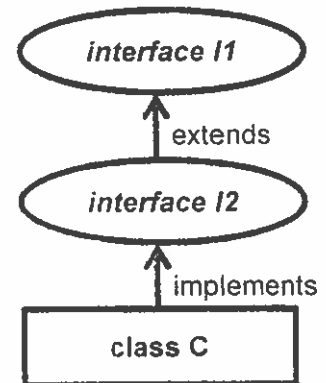
The next 4 questions are multiple choice: circle the *best single response* to each.

2.4. (3 points) Which parameter mode means the outgoing value, in fact the entire behavior of the method, is independent of the incoming value of that parameter?

- ☒ Clears
- ☐ Replaces
- ☐ Restores
- ☐ Updates

2.5. (3 points) The diagram to the right implies that:

- ☒ C is a description of what I2 does, not of how it does it.
- ☐ C implements the behavior/functionality provided in I1.
- ☐ I1 inherits all the methods declared in I2.
- ☐ I2 is the implementer view and I1 is the client view.



2.6. (3 points) According to the “contract” in design-by-contract, a method’s *caller* is permitted to:

- ☒ Assume the precondition is true when calling the method.
- ☐ Assume the precondition is true when the method returns.
- ☐ Assume the method will throw an exception if the precondition is false when calling the method.
- ☐ Assume the postcondition is true when the method returns if the precondition is true when calling the method.

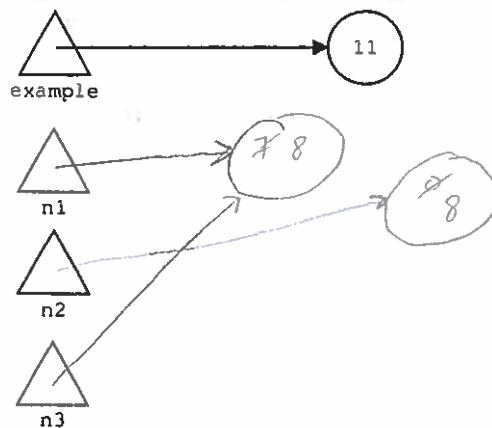
2.7. (3 points) According to the “contract” in design-by-contract, the *implementer* of a method is permitted to:

- ☒ Assume the precondition is true when the method is called.
- ☐ Assume the precondition is true just before the method returns.
- ☐ Throw an exception if the postcondition is not true just before the method returns.
- ☐ Assume the postcondition is true just before the method returns.

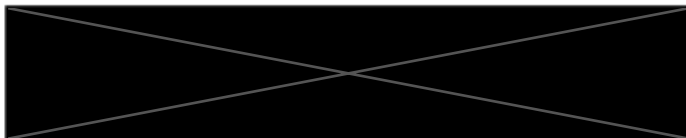
3. (12 points) Consider the following program.

```
public class Problem3 {  
    public static void main(String[] args) {  
        SimpleWriter out = new SimpleWriter1L();  
        NaturalNumber example = new NaturalNumber2(11);  
        NaturalNumber n1 = new NaturalNumber2(7);  
        NaturalNumber n2 = new NaturalNumber2();  
        NaturalNumber n3 = n1;  
        n3.increment();  
        n2.copyFrom(n1);  
        out.println("n1 = " + n1 + ", n2 = " + n2 + ", n3 = " + n3);  
    }  
}
```

3.1. (9 points) Draw a picture tracing the values of each of the *NaturalNumber* references and their corresponding *objects*. Make sure you show *each step of the above program in your diagram*. If reference or object values change, indicate this by crossing off the old value with an X. Do not draw any extra objects when a variable's object value changes but its reference does not. One has been done for you, as an example.



3.2. (3 points) Show *exactly* what the program output is going to be.



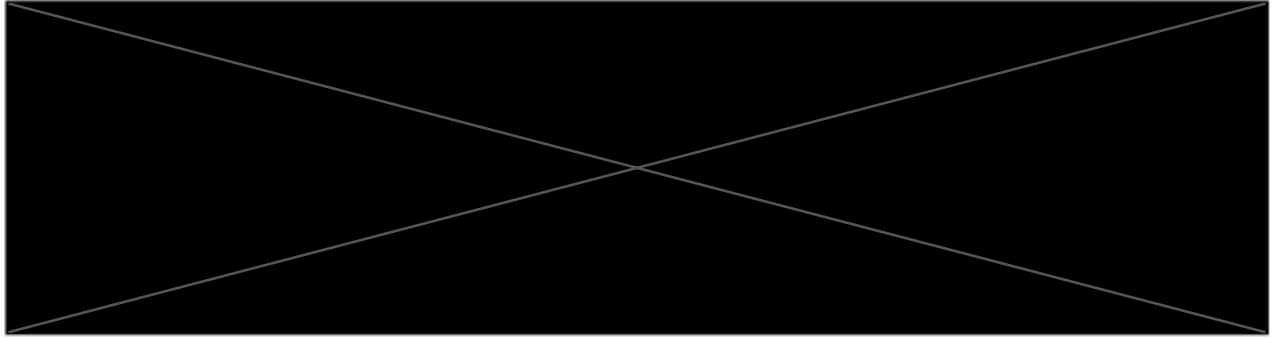
4. (14 points) Tracing (on the next 2 pages)

4.1. (10 points) Complete (as discussed in class) the tracing table on the next page for the given contract and body of the `min` static method, then answer the question after the tracing table. The array values are already filled in for you.

Code	State (Variable Values)
<pre>/** * @requires 0 <= start < nums.length * @ensures min = [the smallest int in nums * between index start and the end of nums] */ private static int min(int[] nums, int start) {</pre>	
	<pre>nums = [2, 7, -1, 3] start = 0</pre>
<pre>int ans = nums[start];</pre>	
	<pre>nums = [2, 7, -1, 3] start = ans =</pre>
<pre>if (start < nums.length - 1) {</pre>	
	<pre>nums = [2, 7, -1, 3] start = ans =</pre>
<pre>int small = min(nums, start + 1);</pre>	
	<pre>nums = [2, 7, -1, 3] start = ans = small =</pre>
<pre>if (small < ans) {</pre>	
	<pre>nums = [2, 7, -1, 3] start = ans = small =</pre>
<pre>ans = small;</pre>	
	<pre>nums = [2, 7, -1, 3] start = ans = small =</pre>
<pre>}</pre>	
	<pre>nums = [2, 7, -1, 3] start = ans =</pre>
<pre>}</pre>	
	<pre>nums = [2, 7, -1, 3] start = ans =</pre>
<pre>return ans;</pre>	
<pre>}</pre>	

- 4.2. (4 points) Based only on the contract (requires and ensures clauses) for `min` provided on the previous page, what is the value of variable `smallest` after the call to method `min` below? Briefly, but convincingly, justify your answer.

```
int[] array = { 1, 2, 3, 4 };  
int smallest = min(array, 4);
```



5. (8 points) Write a method body for the `NaturalNumber` *instance* method `isDivisibleBy5` shown below. The only `NaturalNumber` methods you are allowed to use are the kernel methods `multiplyBy10`, `divideBy10`, and `isZero`. For full credit, your method must use a single `return` statement and it must restore this without making any copies. *No recursion is needed.*

```
/**  
 * Reports whether this is divisible by 5.  
 *  
 * @ensures isDivisibleBy5 = [this is divisible by 5]  
 */  
public boolean isDivisibleBy5() {  
    boolean isDivisible = false;  
    int last = this.divideBy10();  
    if (last == 0 || last == 5) {  
        isDivisible = true;  
    }  
    this.multiplyBy10(last);  
    return isDivisible;  
}
```

6. (12 points) Write a **recursive** method body for the **static** method `digitCount` shown below. The only `NaturalNumber` methods you are allowed to use are the kernel methods `multiplyBy10`, `divideBy10`, and `isZero`. For full credit, your method must use a single **return** statement and it must restore `n` without making any copies.

6

```
/**
 * Reports the number of times digit d appears in number n.
 *
 * @requires 0 <= d <= 9
 * @ensures digitCount = [number of occurrences of digit d in n]
 */
```

```
private static int digitCount(NaturalNumber n, int d) {
```



```
    int count = 0;
```

```
    if (!n.isZero()) {
```

```
        int possibleD = n.divideBy10();
```

```
        if (possibleD == d) {
```

```
            count = digitCount(n, d);
```

```
            count++;
```

```
        }
```

```
    }
```

```
    n.multiplyBy10(possibleD);
```

```
    return count;
```

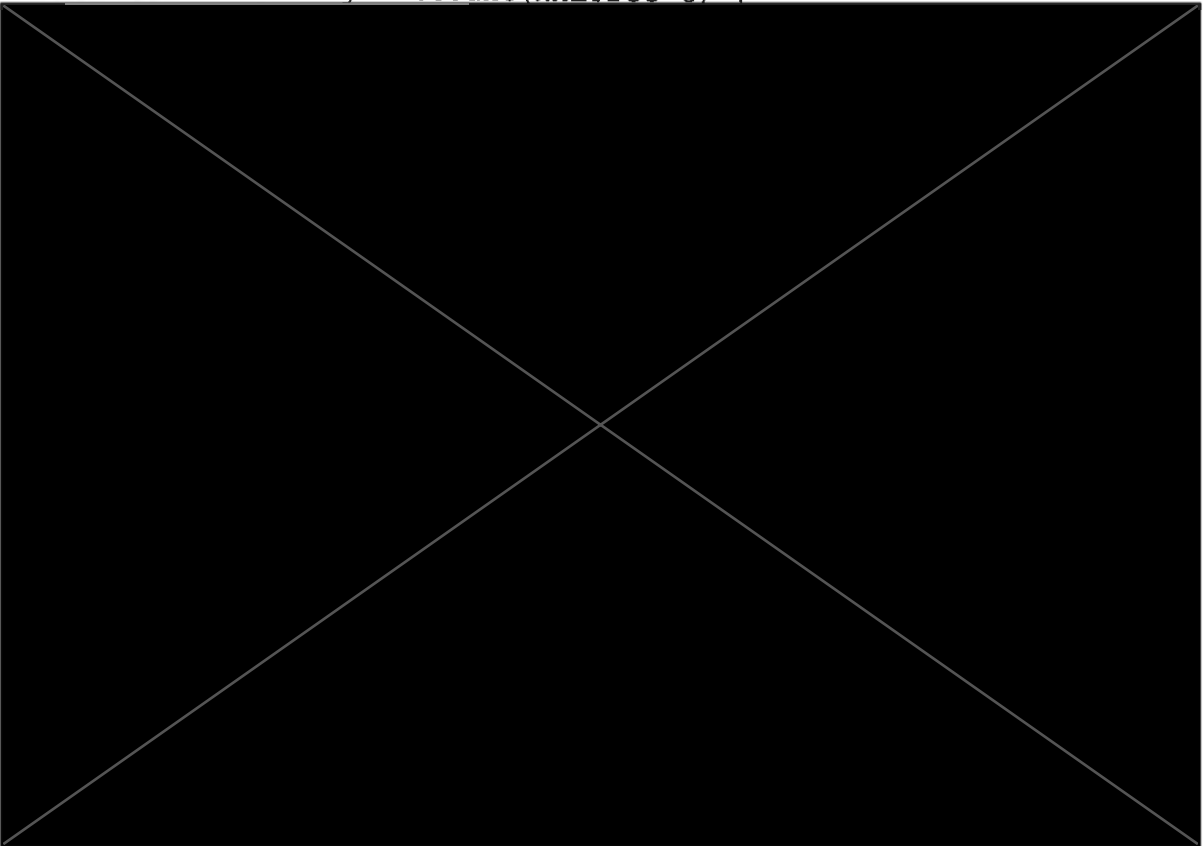
-2

shared out
defeat on
digit being d
-2

7. (12 points) Write a *recursive* method body for the *static* method `tagLeafCount` whose contract is given below. For full credit, your method must use a single `return` statement.

2

```
/**
 * Reports the number of tag nodes in t with no children.
 *
 * @ensures
 *   tagLeafCount = [number of tag nodes in t with no children]
 */
private static int tagLeafCount(XMLTree t) {
```



)

This page may be used as scratch space.

Interface NaturalNumberKernel

```
public interface NaturalNumberKernel
    extends Standard<NaturalNumber>
```

Natural number kernel component with primary methods. (Note: by package-wide convention, all references are non-null.)

Mathematical Subtypes:

NATURAL is integer

exemplar n

constraint $n \geq 0$

Mathematical Model (abstract value and abstract invariant of this):

type NaturalNumberKernel is modeled by NATURAL

Constructor(s) (initial abstract value(s) of this):

():

ensures

this = 0

(int i):

requires

$i \geq 0$

ensures

this = i

(String s):

requires

there exists n : NATURAL ($s = \text{TO_STRING}(n)$)

ensures

$s = \text{TO_STRING}(this)$

(NaturalNumber n):

ensures

this = n

Method Summary

All Methods

Instance Methods

Abstract Methods

Modifier and Type	Method and Description
int	divideBy10() Divides this by 10 and reports the remainder.
boolean	isZero() Reports whether this is zero.
void	multiplyBy10(int k) Multiplies this by 10 and adds k.

Methods inherited from interface components.standard.Standard

clear, newInstance, transferFrom

Interface XMLTree

```
public interface XMLTree
```

XMLTree component with all its methods. (Note: by package-wide convention, all references are non-null.)

An XMLTree is modeled by a tree where each node has a label (either a tag or some text) and if the label is a tag, the node also has a set of (attribute, value) pairs.

There are two constructors. One takes the name of a file or a URL and a boolean flag, and constructs an XMLTree corresponding to the XML read from the file or URL. The boolean flag indicates whether leading/trailing whitespace should be trimmed from text nodes. The second constructor only takes the name of a file or a URL and behaves exactly as if the first constructor was invoked with a true flag (i.e., by default it trims whitespace).

Method Summary

All Methods	Instance Methods	Abstract Methods
Modifier and Type	Method and Description	
Iterable<String>	attributeNames()	Returns an Iterable<String> of the attribute names of the root of this.
String	attributeValue(String name)	Returns the value associated with the attribute of the root tag of this called name.
XMLTree	child(int x)	Returns the x-th subtree of the root of this.
void	display()	Displays this in a new window.
boolean	hasAttribute(String name)	Returns whether the root tag of this has an attribute called name.
boolean	isTag()	Returns whether the label of the root of this is a tag.
String	label()	Returns the label of the root of this.
int	numberOfChildren()	Returns the number of subtrees of the root of this.
String	toString()	Returns the XML string representation of this.