Midterm 2 Review

14. RSS

Example RSS Feed

```
<rss version="2.0">
  <channel>
    <title>Yahoo! News</title>
    <link>http://news.yahoo.com</link>
    <description>The latest news and headlines from Yahoo! News.
    </description>
    <item>
      <title>Apple seeks to stop…</title>
      <link>http://news.yahoo.com/...</link>
      <description>Apple Inc will seek a...</description>
      <pubDate>Mon, 27 Aug 2012 14:40:49</pubDate>
      <source url="http://www.reuters.com">Reuters</source>
    </item>
    ...
  </channel>
</rss>
```

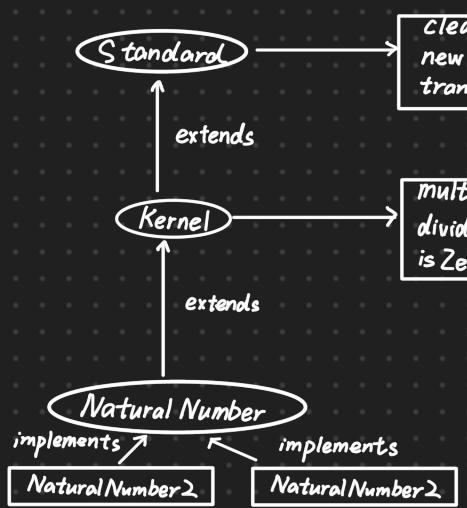
exactly one

< rss >
1 < channel >
< title > } required
< link >
< description >
< item > 0 or more
< title > } at least one title or description
< description >

Mathematical Modulo ("mod")

	mod	%
67,24	19	19
-67,24	5	-19

15. Natural Number



Separating standard methods into their own interface allows these highly reused methods to be described once. → single point of control over change

Kernel method / primary method

Secondary methods

Constructors:
① No-argument Constructor
② Copy Constructor
③ Constructor from int
④ Constructor from string

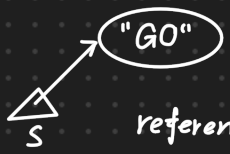
Methods: all the methods are instance method.

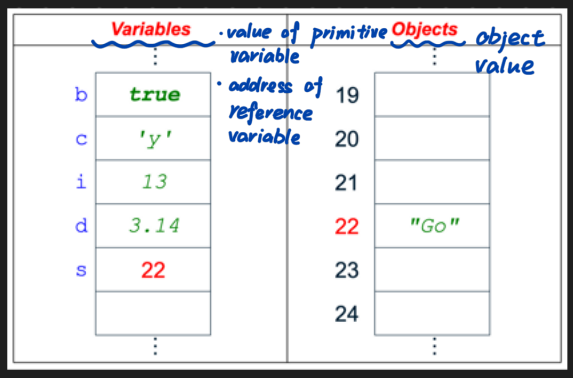
add()
subtract()
multiply()
divide()
power()
root()

copyFrom()
compareTo()
multiplyBy10()
divideBy10()
isZero()
clear()

newInstance()
return negative int if this < n
m.transferFrom(n) m = #n, n = 0
toString()

16. Reference



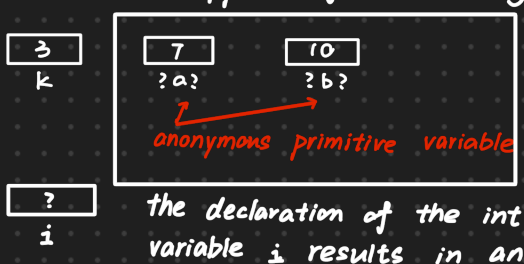


Value : ① reference value : memory address

② object value : "GO"

* in tracing table, use s → "Go"

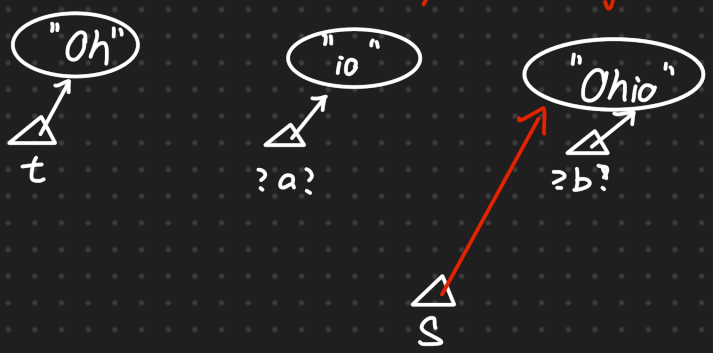
Assignment for Primitive Types : int k = 3;
int i = k+7;



the declaration of the int variable i results in an uninitialized primitive variable

Assignment for Reference Type : String t = "Oh";
String s = t + "io";

anonymous reference type



① String concatenation operator + result in ?b?

② s is uninitialized reference variable

③ assignment operator copies the reference value into s

④ Temporary reference variable disapper, but objects "io" remain

⑤ "io" into the garbage collector

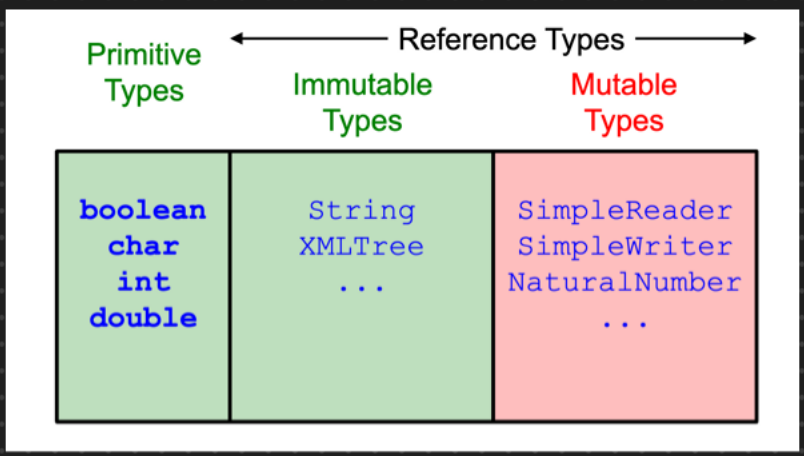
Alias

A Tracing Table Using ➡

Code	State
	`t ➡ "Oh"`
`String s = t;`	
	`s,t ➡ "Oh"`

Why It Matters: `NaturalNumber`

Code	State
	`z ➡ 99`
`NaturalNumber n = z;`	
	`z,n ➡ 99`
`n.increment();`	
	`z,n ➡ 100`



parameter passing for references:

- method calls copy reference type
- return value of the method and copy back the reference type

Equality Checking: ① == compares reference value

② .equals() compare object value

for NN, .equals can do both

17. Arrays and References

1. Overview: ① Each Element references one of these variables

② a.length

2. Arrays as Reference Type: the name of array refers { entire collection / a.length

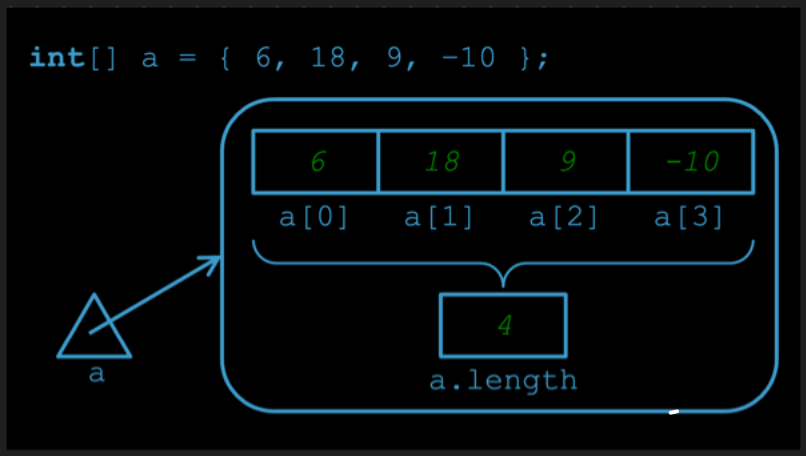
① Assign one another using = : copy reference value

② Pass an array as parameter to a method : reference value

③ Return an array : return reference value

④ Compare using == : compare reference value

⑤ Compare using .equals() : equivalent to ==



How to compare : ① Using own code

② import java.util.Arrays

Arrays.equals(arr1, arr2)

18. Contracts

including - Parameter Modes

- Two stipulations : ① Parameter names in requires and ensures clauses stand for object values

② Reference-type arguments are always non-null

Parameter Modes:

① Restores Mode : - A restores-mode parameter once again has its incoming value upon return from a method call

- default parameter mode, if a parameter is not listed with other modes then its mode is restores

- Equivalent to adding .... and x = [the initial value of x] to the ensures clause

- An old restores mode para.. #x shouldn't appear in the ensures clause

② Clears Mode :

- Upon return, a clears-mode parameter has an initial value for its type.
- Ex) a no-argument constructor
  But it's possible there is no no-argument constructor
- a clears-mode parameter x should not appear in ensures clause

③ Replaces Mode

- a replaces-mode parameter has a value that might be changed from its incoming value, but the method's behavior doesn't depend on its incoming value
- a replace-mode parameter x should not appear in the requires clause
  # x should not in the ensures clause

④ Updates Mode

- a updates-mode parameter has a value change. the method's behavior depend on incoming value

Null Reference :

String s = null;

NullPointerException

19. Repeated Arguments

Source of Aliasing : ① Simple Assignment =

② Parameter Passing : formal parameter is initialized by copying argument's reference value

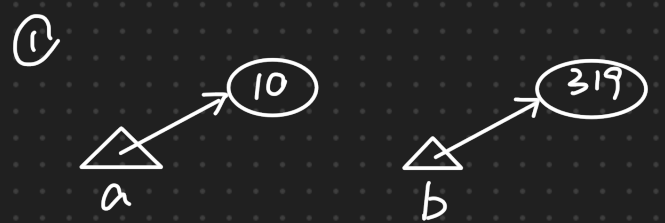
Example

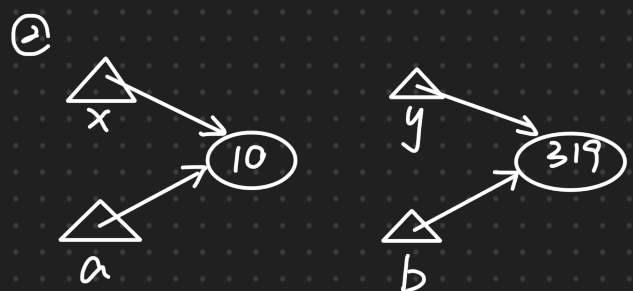
- Consider this method:

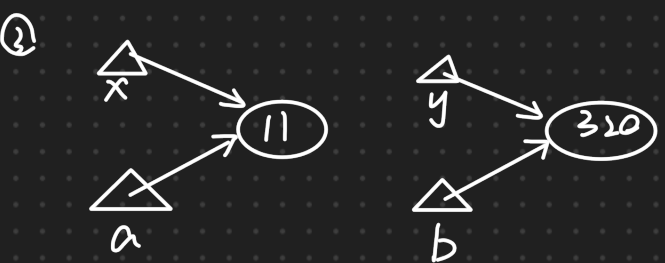
```
/**
* Adds 1 to the first number and 2 to the
* second.
* ...
* @updates x, y
* @ensures
* x = #x + 1  and  y = #y + 2
*/
private static void foo(NaturalNumber x,
    NaturalNumber y) {...}
```

Consider this call of the method:

```
NaturalNumber a = new NaturalNumber2(10);
NaturalNumber b = new NaturalNumber2(319);
foo(a, b);
```







Harmless Aliasing : a, b are not in scope

- Now consider this call of the method:

```
NaturalNumber a = new NaturalNumber2(10);
foo(a, a);
```

Outcome : not 13

③ The Receiver is an Argument :

n.add(n)

Never Pass any variable of a mutable reference type as an argument twice or more to a single method call

+ Receiver is an argument

Interval - Hazing

```
private static int root(int n, int r)
{
        int lowEnough = 0;
        int tooHigh = n+1;
        while(lowEnough+1 <tooHigh)
        {
                int guess = (lowEnough+tooHigh)/2;
                if (Math.power(guess,r)<=n){
                        lowEnough = guess;
                }else{
                        tooHigh = guess;
                }
        )
        return LowEnough;
}
```

```
public static void root(NaturalNumber n, int r) {
    assert n != null : "Violation of: n is  not null";
    assert r >= 2 : "Violation of: r >= 2";

    NaturalNumber tooLow = new NaturalNumber2(0);
    NaturalNumber tooHigh = new NaturalNumber2();
    tooHigh.copyFrom(n);
    tooHigh.increment();
    NaturalNumber guess = new NaturalNumber2(0);
    NaturalNumber two = new NaturalNumber2(2);
    NaturalNumber tooLowPlusOne = new NaturalNumber2(1);

    while (tooLowPlusOne.compareTo(tooHigh) < 0) {
        guess.copyFrom(tooLow);
        guess.add(tooHigh);
        guess.divide(two);
        NaturalNumber power = new NaturalNumber2();
        power.copyFrom(guess);
        power.power(r);
        if (power.compareTo(n) > 0) {
            tooHigh.copyFrom(guess);
        }
        if (power.compareTo(n) < 0) {
            tooLow.copyFrom(guess);
        }
        if (power.compareTo(n) == 0) {
            tooHigh.copyFrom(guess);
            tooLow.copyFrom(guess);
        }
        tooLowPlusOne.copyFrom(tooLow);
        tooLowPlusOne.increment();
    }
    n.copyFrom(tooLow);
}
```

20. Mathematical String Notation

T : entry type

string of T

1. Math Notation :

① Empty string : < > / empty-string

② string : <1, 2, 3>

<'G', 'O'> → String

< > → or "GO"

③ Concatenation : s * t

<1, 2> * <3, 4> = <1, 2, 3, 4>

"GO" * "" = "GO"

2. substring : s[i, j)

prefix 前缀 suffix 后缀

3. length : |<1, 2, 3, 4>| = 4

|< >| = 0

4. Reverse : rev(s)

5. Permutation : perms(<1, 2>, <2, 1>)

not perms(<1, 2>, <3, 4>)

6. Occourence Count : count(<1, 2, 3, 2, 2>, 2) = 3

21. Recursion Thinking

```
private static void increment (NaturalNumber n) {
  int onesDigit = n.divideBy10();
  onesDigit++;
  if (onesDigit == 10) {
    onesDigit = 0;
    increment(n);
  }
  n.multiplyBy10(onesDigit);
```

```
@Override
public void power(int p) {
    assert p >= 0 : "Violation of: p >= 0";
    NaturalNumber temp = new NaturalNumber2();
    temp.copyFrom(this);
    if (p == 0) {
        this.setFromInt(1);
    } else if (p % 2 == 1 && p > 2) {
        this.power(p - 1);
        this.multiply(temp);
    } else if (p % 2 == 0) {
        this.power(p / 2);
        NaturalNumber temp2 = new NaturalNumber2();
        temp2.copyFrom(this);
        this.multiply(temp2);
    }
}
```

22. Concepts of Object-Oriented Programming

1. Implement : between class and interface

If C implements I , - C has the method body for instance method whose contracts are specified in I

- ```
  class C implements I {
    // bodies for methods specified in I
  }
  ```

Allow separate contracts from their implementations — the best practise for component design

2. Extend : hold between Two interfaces / Two classes

If B extends A, B inherits all the methods of A

3. Caveats about Java Interface :

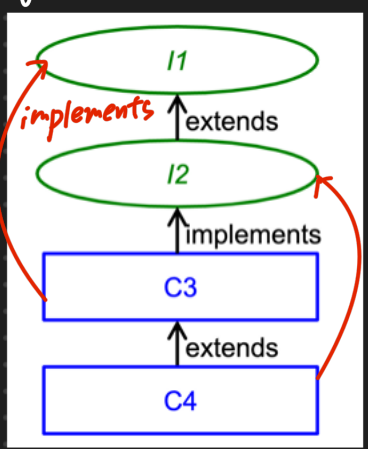
① Interface cannot have constructor

② Interface cannot have static method contract without providing corresponding method bodies, so there is no good place to write separate contract for public static method

③ Constructors are not inherited, but bodies of constructors in B can invoke the constructor of A, by super(...)

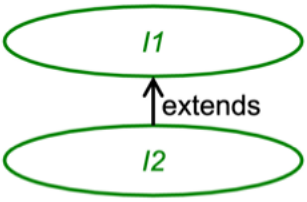
④ Static method are inherited

4. Interface Extension




```
interface I2 extends I1 {
  // contracts for methods added in I2
}
```

instance method



I_2 is a subinterface of I_1
derived interface
child interface

I_1 is a superinterface of I_2
base interface
parent interface
```

5. Class Extension ① add method bodies
② override by providing new method bodies

> Important note: **Overriding** a method is different from **overloading** a method! A method (name) is **overloaded** when two or more methods have the same name, in which case the methods must differ in the number and/or types of their formal parameters (which the compiler uses to disambiguate them).

6. Declared Type: when a variable is declareed using the name of interface as its type, then its declared type is said to be an interface type

when a variable is declareed using the name of class as its type, then its declared type is said to be an class type

7. Object Type. when a variable is instantiated then its object type / dynamic type is the type from which the constructor comes.

```
NaturalNumber k =
  new NaturalNumber2();
```

* If declared type is I, Object type is C, then C implements I must hold

8. Polymorphism

java decide which method body to use for any call to an instance method based on the object type of receiver

Static & Instance Method

Both: ① May have formal parameters
② May return any type
③ May be public or private

25. Testing

unit testing : test individual unit

Testing: a technique for trying to refute the claim that a method body is correct for the method contract

Testing : find defect
vs
Debugging : find defect and repair it

26. JUnit