

## Chapter 4

### Array

#### 4.1 Introduction

#### 4.2 Various array's supported in Java

## 4.2. VARIOUS ARRAY'S SUPPORTED IN JAVA

```
class Array{
    private static final IntUtil u = new IntUtil();

    private static void testStaticArray() {
        int [] a = {10,20,30,50} ;
        int l = a.length ;
        u.pLn("Before ",a,0,l);
        a[2] = -90 ;
        u.pLn("After ",a,0,l);
        //a[5] = 67 ;
        //Exception in thread "main"
        //java.lang.ArrayIndexOutOfBoundsException: 5
    }
}
```

**1**

- 1. Size should be known at compilation
- 2. Cannot change size

```
private static void testDynamicArray(int s) {
    int a[] = new int[s];
    int l = a.length ;
    assert(s == l);
    a[8] = 8 ;
    a[7] = 78 ;
    a[2] = 47 ;

    u.pLn(l); ;
    u.pLn(a,0,l);
    //You cannot delete a[3] ;
    //a[s+5] = 25 ;
    //java.lang.ArrayIndexOutOfBoundsException: 15
}
}
```

**2**

- 1. Size should be known at run time
- 2. Cannot change size

```
private static void testintArrayList(){
    //ArrayList<int> a = new ArrayList<int>() ;
    //ArrayList can only reference types, not primitives.
}
}
```

**3**

Cannot insert primitive types

Figure 4.1: Various array's supported by Java

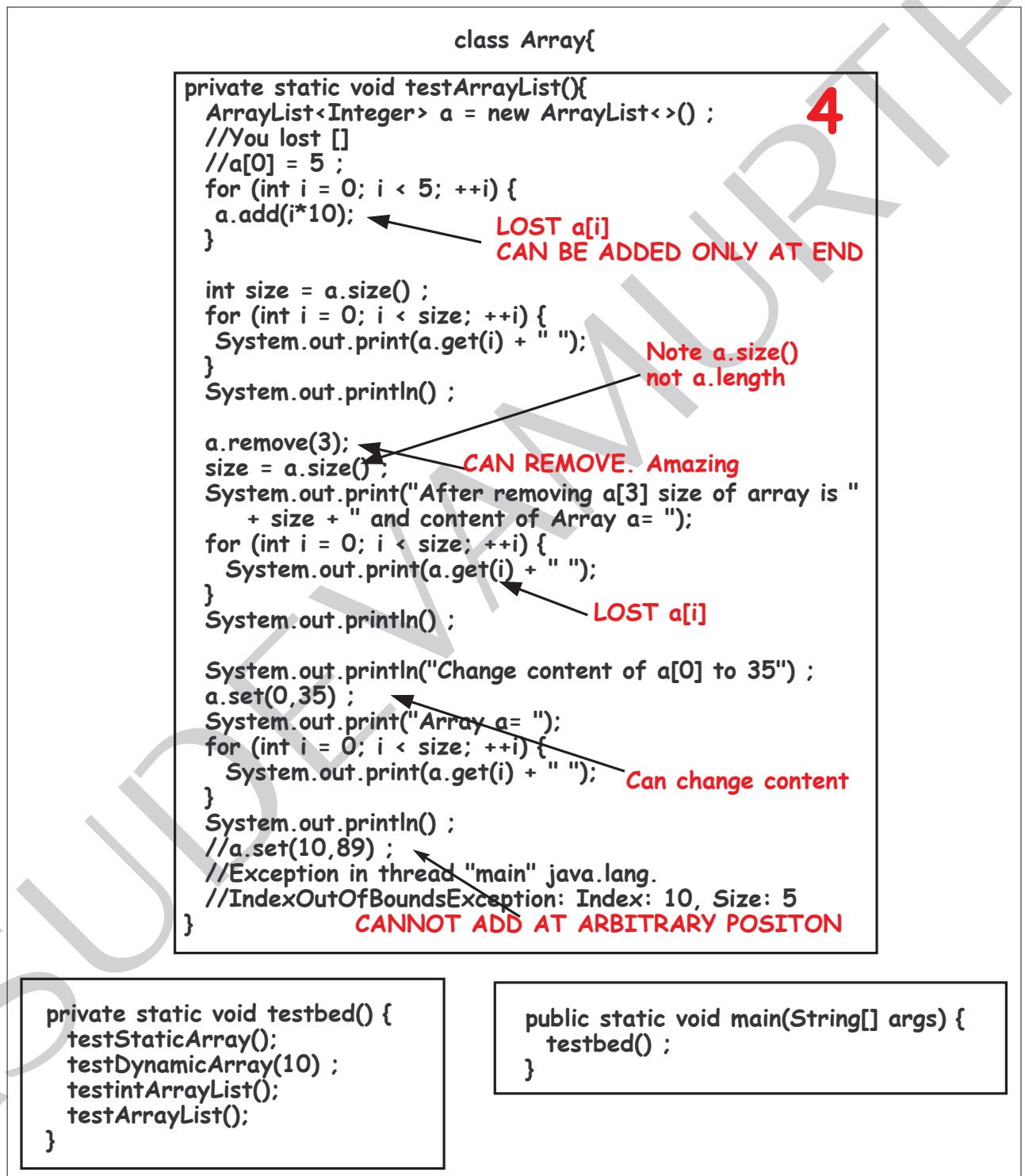


Figure 4.2: Various array's supported by Java

### 4.3 Dynamic growable int array

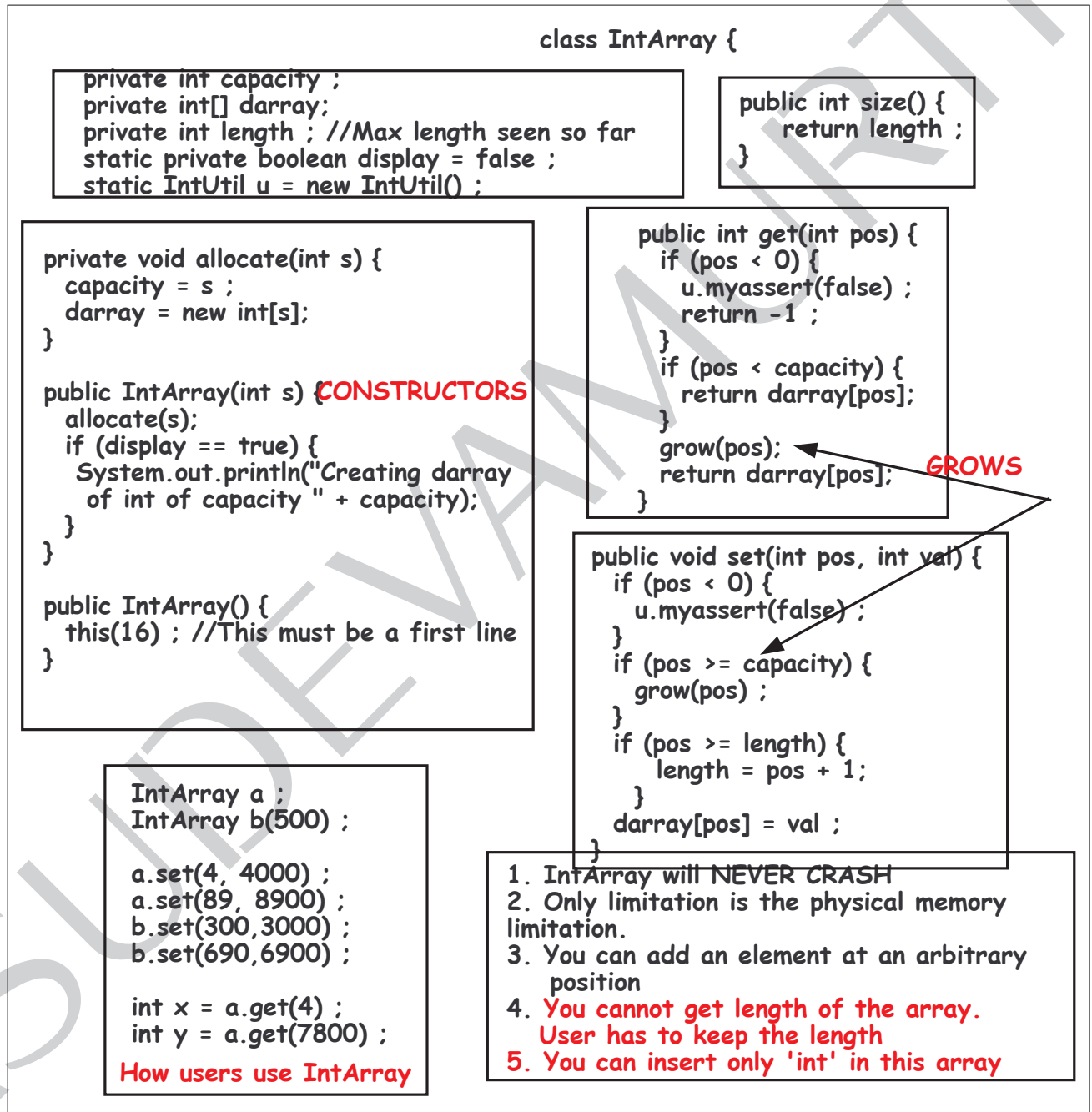


Figure 4.3: An int array that can grow

#### 4.3. DYNAMIC GROWABLE INT ARRAY

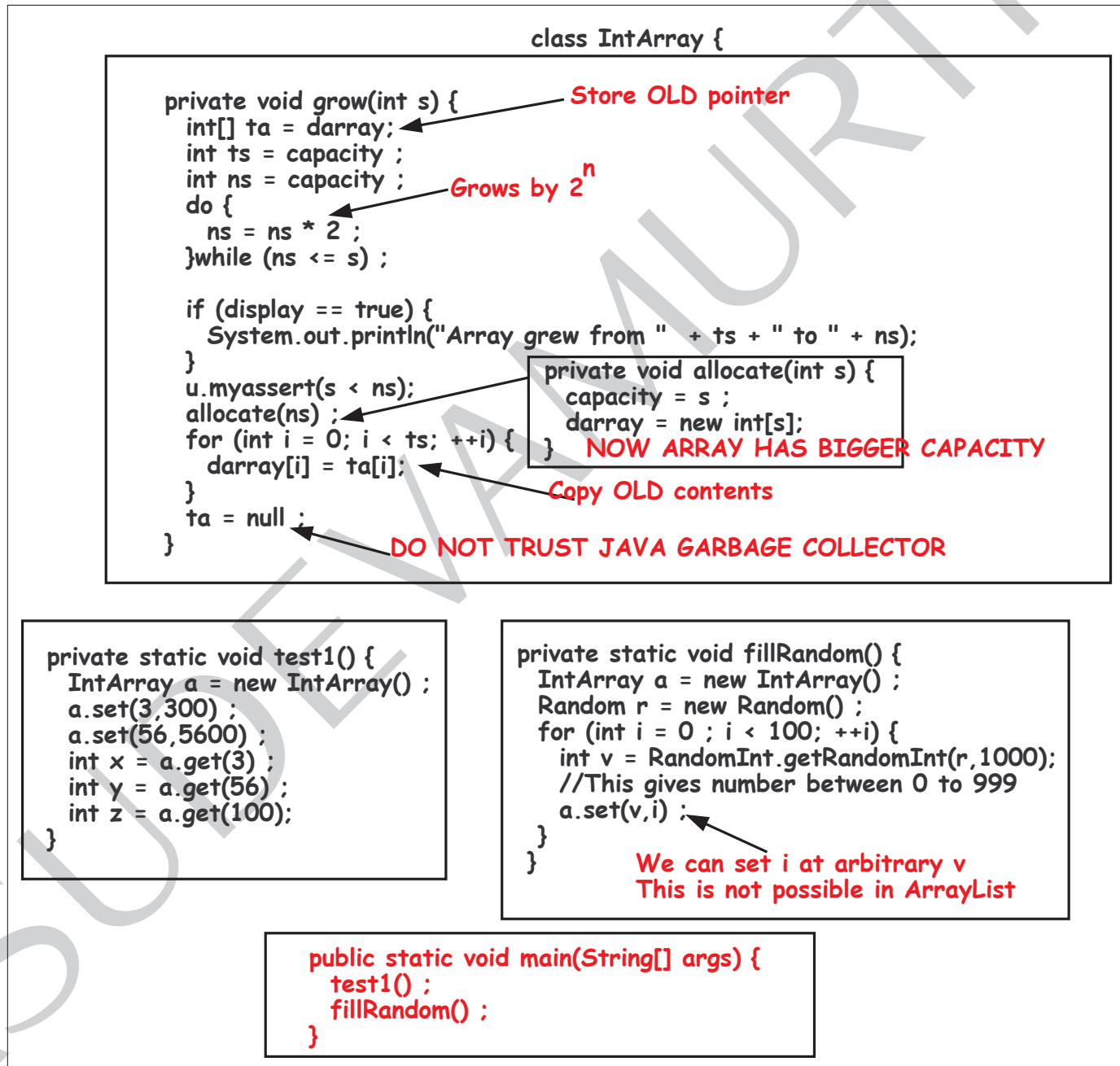


Figure 4.4: An int array that can grow

## 4.4 Dynamic growable char array

```

class CharArray {
    private int capacity;
    private char[] darray;
    static private boolean display = false;
    static IntUtil u = new IntUtil();

    private void allocate(int s) {
        capacity = s;
        darray = new char[s];
    }

    public CharArray(int s) {
        allocate(s);
    }

    public CharArray() {
        this(16);
    }

    private void grow(int s) {
        char[] ta = darray;
        int ts = capacity;
        int ns = capacity;
        do {
            ns = ns * 2;
        } while (ns <= s);
        u.myassert(s < ns);
        allocate(ns);
        for (int i = 0; i < ts; ++i) {
            darray[i] = ta[i];
        }
        ta = null;
    }

    public char get(int pos) {
        if (pos < 0) {
            u.myassert(false);
        }
        if (pos < capacity) {
            return darray[pos];
        }
        grow(pos);
        return darray[pos];
    }

    public void set(int pos, char val) {
        if (pos < 0) {
            u.myassert(false);
        }
        if (pos >= capacity) {
            grow(pos);
        }
        darray[pos] = val;
    }
}

```

1. Never crashes
2. Only limitation is physical memory
3. Can add a character in arbitrary position
4. No length concept. User needs to keep
5. Only char can be inserted

Figure 4.5: A char array that can grow

## 4.5. C STRING

```
private static void test1() {  
    CharArray b = new CharArray();  
    int s = 0 ;  
    for (int i = 0; i < 8; ++i) {  
        b.set(i, (char)('a'+i));  
        ++s ;  
    }  
    b.pln("from 0 to " + (s-1) + ": ", 0, s-1);  
    b.pln("from " + (s-1) + " to 0: ", s-1, 0);  
    CharArray a = new CharArray();  
    a.set(3, 'Z');  
    a.set(56, 'U');  
    char x = a.get(3);  
    char y = a.get(56);  
    char z = a.get(100);  
    System.out.println("a[3]= " + x + " a[56] = " + y + " a[100] = " + z);  
}
```

Creating darray of int of capacity 16  
from 0 to 7: abcdefgh  
from 7 to 0: hgfedcba

Creating darray of int of capacity 16  
Array grew from 16 to 64  
Array grew from 64 to 128  
a[3]=Z a[56]=U a[100]=  
Null char '\0

Figure 4.6: Using class *CharArray*

## 4.5 C string



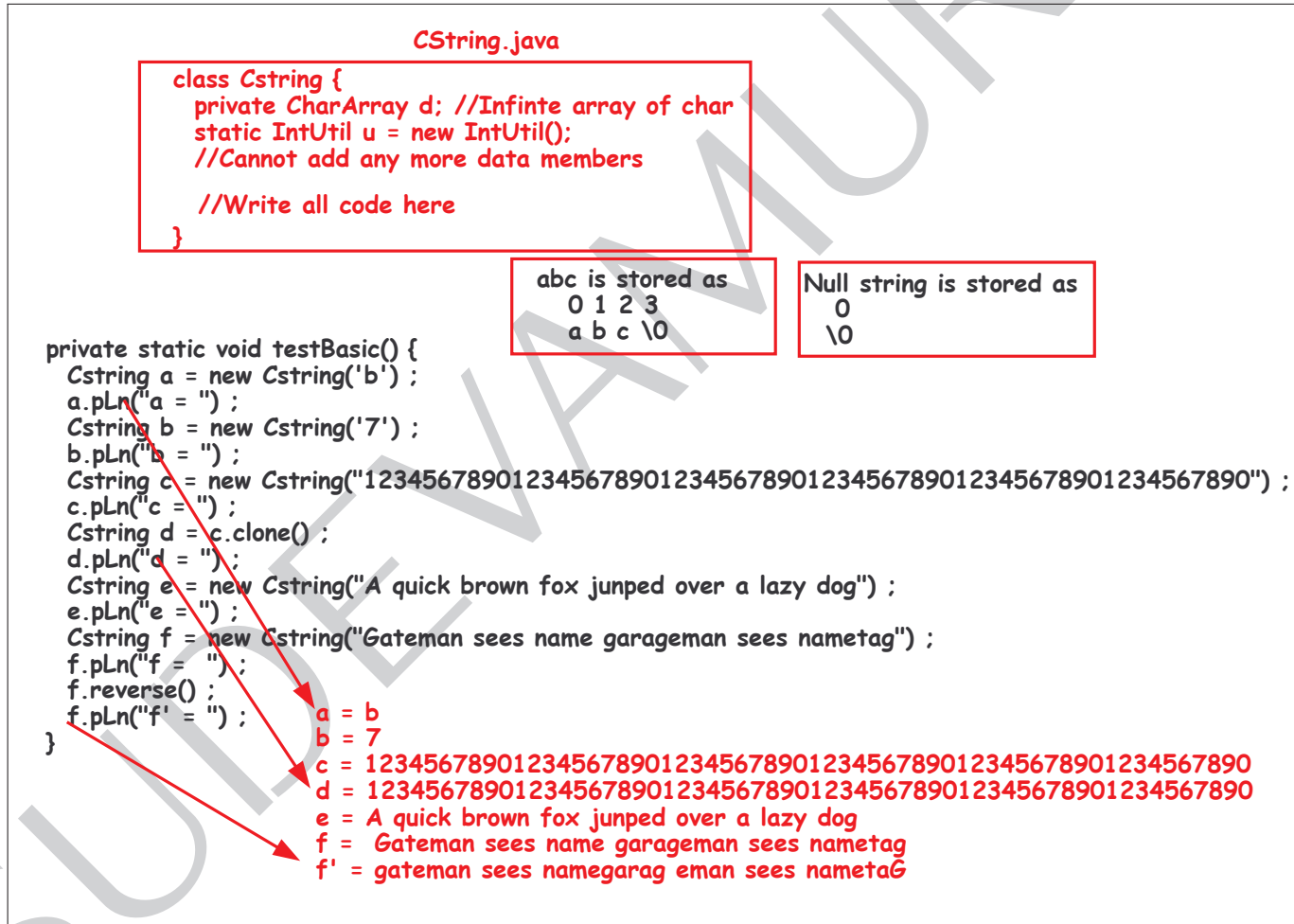


Figure 4.7: Mutable CString class

## 4.5. C STRING

**CString.java**

```
private static void testAdd() {  
    CString a = new CString("UCSC") ;  
    CString b = new CString("Extension") ;  
    CString c = a.add(b) ;  
    a.pLn("a = ") ;  
    b.pLn("b = ") ;  
    c.pLn("c = ") ;  
    CString d = c.add("USA") ;  
    d.pLn("d = ") ;  
    a.append(b) ;  
    a.pLn("a+b = ") ;  
    a.append("World") ;  
    a.pLn("a+b+World = ") ;  
}
```

write add and append

a = UCSC  
b = Extension  
c = UCSCExtension  
d = UCSCExtensionUSA  
a+b = UCSCExtension  
a+b+World = UCSCExtensionWorld

mutable

```
private static void testEqual() {  
    CString a = new CString("1234567890123456789012345678901234567890123456789012345678901234567890") ;  
    a.pLn("a = ") ;  
    CString b = new CString("1234567890123456789012345678901234567890123456789012345678901234567890") ;  
    b.pLn("b = ") ;  
    u.myassert(a.isEqual(b)) ;  
    CString c = new CString("1234567890123456789012345678901234567890123456789012345678901234567890") ;  
    c.pLn("c = ") ;  
    u.myassert(a.isEqual(c) == false) ;  
}
```

write isEqual

Figure 4.8: *add append* and *isEqual* on *Cstring*

[illegible]

### 4.6.1 Addition of big Number

## 4.6. BIG NUMBER

**BigNumberTester.java**

```
private static void testAdd() {
    BigInteger a = new BigInteger("9789");
    BigInteger b = new BigInteger("100000");
    a.println("a = ");
    b.println("b = ");
    BigInteger c = a.add(b);
    c.println("a + b = c = ");
    a = a.add(b);
    a.println("a = a + b = ");
    BigInteger d = new BigInteger("3490529510847650949147849619903898133417764638493387843990820577");
    BigInteger e = new BigInteger("32769132993266709549961988190834461413177642967992942539798288533");
    BigInteger f = d.add(e);
    BigInteger g = new BigInteger("36259662504114360499109837810738359546595407606486330383789109110");
    u.myassert(f.isEqual(g));
    d.println("d = ");
    e.println("e = ");
    f.println("f = ");
}
```

**Writing add in BigInteger class**

**Cannot pull any function from java library  
Implement school addition**

**Must match with known answer in g**

a = 9789  
b = 100000  
a + b = c = 109789  
a = a + b = 109789  
d = 3490529510847650949147849619903898133417764638493387843990820577  
e = 32769132993266709549961988190834461413177642967992942539798288533  
f = 36259662504114360499109837810738359546595407606486330383789109110

Figure 4.10: Writing *add* in BigInteger class

### 4.6.2 Subtraction of big Number

**BigUnsignedNumberTester.java**

Writing sub in BigUnsignedNumber class

```

private static void subtract1(String t, String a1, String b1, String e1){
    BigInteger a = new BigInteger(a1);
    BigInteger b = new BigInteger(b1);
    BigInteger c = a.sub(b);
    u.myassert(c.isEqual(e1));
}

```

Write subtraction using school method

```

private static void rsaSubTests() {
{
    String a = new String("3490529510847650949147849619903898133417764638493387843990820577");
    String b = new String("32769132993266709549961988190834461413177642967992942539798288533");
    String e = new String("-29278603482419058600814138570930563279759878329499554695807467956");
    subtract1("RSA", a, b, e);
}
    c = -29278603482419058600814138570930563279759878329499554695807467956
{
    String a = new String("3490529510847650949147849619903898133417764638493387843990820577");
    String b = new String("32769132993266709549961988190834461413177642967992942539798288533");
    String e = new String("29278603482419058600814138570930563279759878329499554695807467956");
    subtract1("RSA", b, a, e);
}
    c = 29278603482419058600814138570930563279759878329499554695807467956
}

```

Figure 4.11: Writing *sub* in BigInteger class

### 4.6.3 Multiplication of big Number

## 4.6. BIG NUMBER

**BigIntegerTester.java**

```
private static void testMult() {
{
    BigInteger a = new BigInteger(9789) ;
    BigInteger zero = new BigInteger(0) ;
    BigInteger one = new BigInteger(1) ;
    a.println("a = ") ;
    zero.println("zero = ") ;
    one.println("one = ") ;
    BigInteger b = a.mult(zero) ;
    b.println("b = a*0 =") ;
    BigInteger c = zero.mult(a) ;
    c.println("c = 0*a = ") ;
    BigInteger d = a.mult(one) ;
    d.println("d = a*1 = ") ;
    BigInteger e = one.mult(a) ;
    e.println("e = 1*a =") ;
}

{
    BigInteger a = new BigInteger(9789) ;
    BigInteger b = new BigInteger(9) ;
    BigInteger c = a.mult(b) ;
    a.println("a = ") ;
    b.println("b = ") ;
    c.println("c = ") ;
    u.myassert(c.isEqual("88101"));
    BigInteger d = b.mult(a) ;
    d.println("d = ") ;
    u.myassert(d.isEqual("88101"));
}
}
```

**Writing mult in BigInteger class**

**Cannot pull any function from java library  
Implement school multiplication**

a = 9789  
zero = 0  
one = 1  
b = a\*0 = 0  
c = 0\*a = 0  
d = a\*1 = 9789  
e = 1\*a = 9789  
a = 9789  
b = 9

**MUST NOT STORE  
PARTIAL PRODUCT**

**C=**

a = 9789  
b = 9  
c = 88101  
d = 88101

**c = a \* b  
multiply two numbers**

1. This routine must use only (a.size+b.size) space
2. There is no need to initialize c to 0
3. You should not use any storage for partial product

**Handwritten Multiplication:**

a	*	b
5 6 7 8	*	9 1 0 5
-----		
2 8 3 9 0		
0 0 0 0		
5 6 7 8		
-----		
5 1 1 0 2		
-----		
5 1 6 9 8 1 9 0		

Figure 4.12: Writing *mult* in BigInteger class

### BigNumberTester.java

#### Writing mult in BigNumber class

```
private static void testMult() {
```

```
{
    BigNumber a = new BigNumber("3490529510847650949147849619903898133417764638493387843990820577");
    BigNumber b = new BigNumber("32769132993266709549961988190834461413177642967992942539798288533");
    BigNumber c = new BigNumber("11438162575788867669235779976146612010218296721242362562561842935706935245733897830597123563958705058989075147599290026879543541");
    BigNumber d = b.mult(a);
    a.println("a = ");
    b.println("b = ");
    c.println("c = ");
    d.println("d = ");
    u.myassert(d.isEqual(c));
}
```

```
a = 3490529510847650949147849619903898133417764638493387843990820577
b = 32769132993266709549961988190834461413177642967992942539798288533
c = 11438162575788867669235779976146612010218296721242362562561
8429357069352457338978305971235639587050589890751475992900268
79543541
d = 11438162575788867669235779976146612010218296721242362562561
8429357069352457338978305971235639587050589890751475992900268
79543541
```

Number of digits in a b and d = 64 65 129

```
private static void testRandom() {
```

```
    int m = 1000;
```

```
    int max = (1 << 15);
```

```
    Random r = new Random();
```

```
    for (int i = 0; i < m; ++i) {
```

```
        //System.out.println("i = " + i);
```

```
        int a = RandomInt.getRandomInt(r, 0, max);
```

```
        int b = RandomInt.getRandomInt(r, 0, max);
```

```
        BigNumber ba = new BigNumber(a);
```

```
        BigNumber bb = new BigNumber(b);
```

```
        BigNumber ma = ba.add(bb);
```

```
        u.myassert(ma.isEqual(a+b));
```

```
        BigNumber mm = ba.mult(bb);
```

```
        u.myassert(mm.isEqual(a*b));
```

```
    }
```

```
    System.out.println("Random addition and multiplication on " + m + " numbers passed");
```

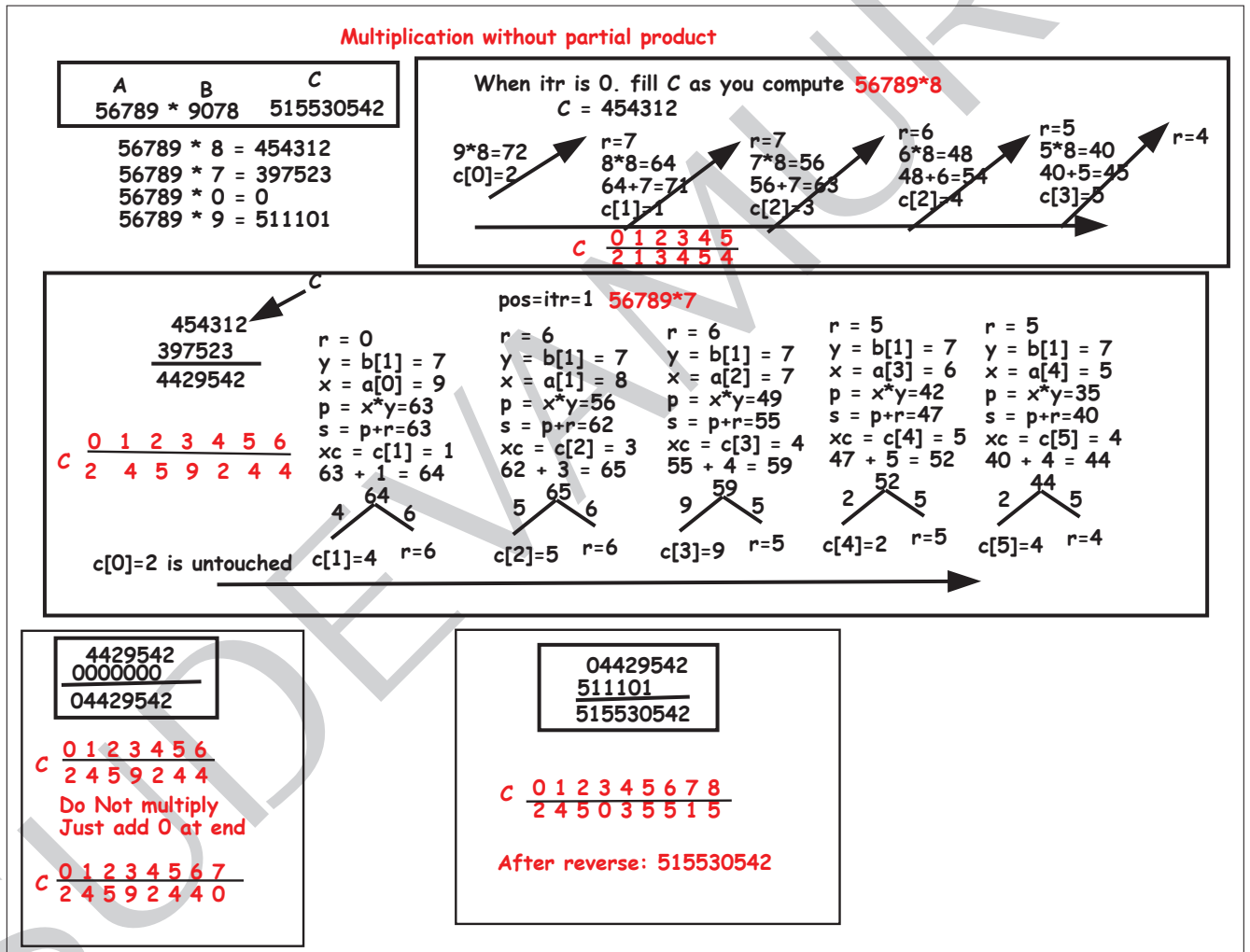
```
}
```

Random addition and multiplication on 1000 numbers passed

All asserts must pass

Figure 4.13: Multiplying a 64 digit number with a 65 digit number

## 4.6. BIG NUMBER



4429542
0000000
04429542

C 0 1 2 3 4 5 6  
2 4 5 9 2 4 4

Do Not multiply  
Just add 0 at end

C 0 1 2 3 4 5 6 7  
2 4 5 9 2 4 4 0

04429542
511101
515530542

C 0 1 2 3 4 5 6 7 8  
2 4 5 0 3 5 5 1 5

After reverse: 515530542

Figure 4.14: Multiplication without storing partial products





#### 4.7. COMPUTING FACTORIAL OF A NUMBER

```
Run time for !1000 = 0.199138015 secs
Number of digits in !1000 = 2568
b = 402387260077093773543702433923003985719374864210714632543799
9104299385123986290205920442084869694048004799886101971960586
3166687299480855890132382966994459099742450408707375991882362
7727188732519779505950995276120874975462497043601418278094646
4962910563938874378864873371191810458257836478499770124766328
8983595573543251318532395846307555740911426241747434934755342
8646576611667797396668820291207379143853719588249808126867838
3745597317461360853795345242215865932019280908782973084313928
4440328123155861103697680135730421616874760967587134831202547
8589320767169132448426236131412508780208000261683151027341827
9777047846358681701643650241536913982812648102130927612448963
5992870511496497541990934222156683257208082133318611681155361
5836546984046708975602900950537616475847728421889679646244945
1607653534081989013854424879849599533191017233555566021394503
9973628075013783761530712776192684903435262520001588853514733
1611702103968175921510907788019393178114194545257223865541461
0628921879602238389714760885062768629671466746975629112340824
3920816015378088989396451826324367161676217916890977991190375
4031274622289988005195444414282012187361745992642956581746628
3029555702990243241531816172104658320367869061172601587835207
5151628422554026517048330422614397428693306169089796848259012
5458327168226458066526769958652682272807075781391858178889652
2081643483448259932660433676601769996128318607883861502794659
5513115655203609398818061213855860030143569452722420634463179
7460594682573103790084024432438465657245014402821885252470935
1906209290231364932734975655139587205596542287497740114133469
6271542284586237738753823048386568897646192738381490014076731
044664025989949022221765904339901886018566526485061799702356
1938970178600408118897299183110211712298459016419210688843871
2185564612496079872290851929681937238864261483965738229112312
5024186649353143970137428531926649875337218940694281434118520
1580141233448280150513996942901534830776445690990731524332782
8826986460278986432113908350621709500259738986355427719674282
2248757586765752344220207573630569498825087968928162753848863
3969099598262809561214509948717012445164612603790293091208890
8694202851064018215439945715680594187274899809425474217358240
1063677404595741785160829230135358081840096996372524230560855
9037006242712434169090041536901059339838357779394109700277534
72000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000
0000000
```

Figure 4.16: Factorial 1000

## 4.8 Problem set

**Problem 4.8.1.** Implement *Cstring* class and *BigNumber* as explained in figure 4.17

1. *CharArray.java* (Nothing has to be changed in this file. Use as is)
2. *Cstring.java*

```
private CharArray d; //Infinite array of char
static IntUtil u = new IntUtil();
```

Cannot add any more data fields  
Write all functions so that all tests will pass

3. *BigNumber.java*
- ```
private Cstring d; //data
static IntUtil u = new IntUtil();
```

Cannot add any more data fields  
Write all functions so that all tests will pass

4. *BigNumberTest.java*
- Nothing has to be changed in this file. Use as is.  
All tests must pass

email: 1. *Cstring.java*  
2. *BigNumber.java*  
3. Output captured in a text file

Figure 4.17: Implementing *Cstring* class and *BigNumber* class