Chapter 4

Array

4.1 Introduction

4.2 Various array's supported in Java

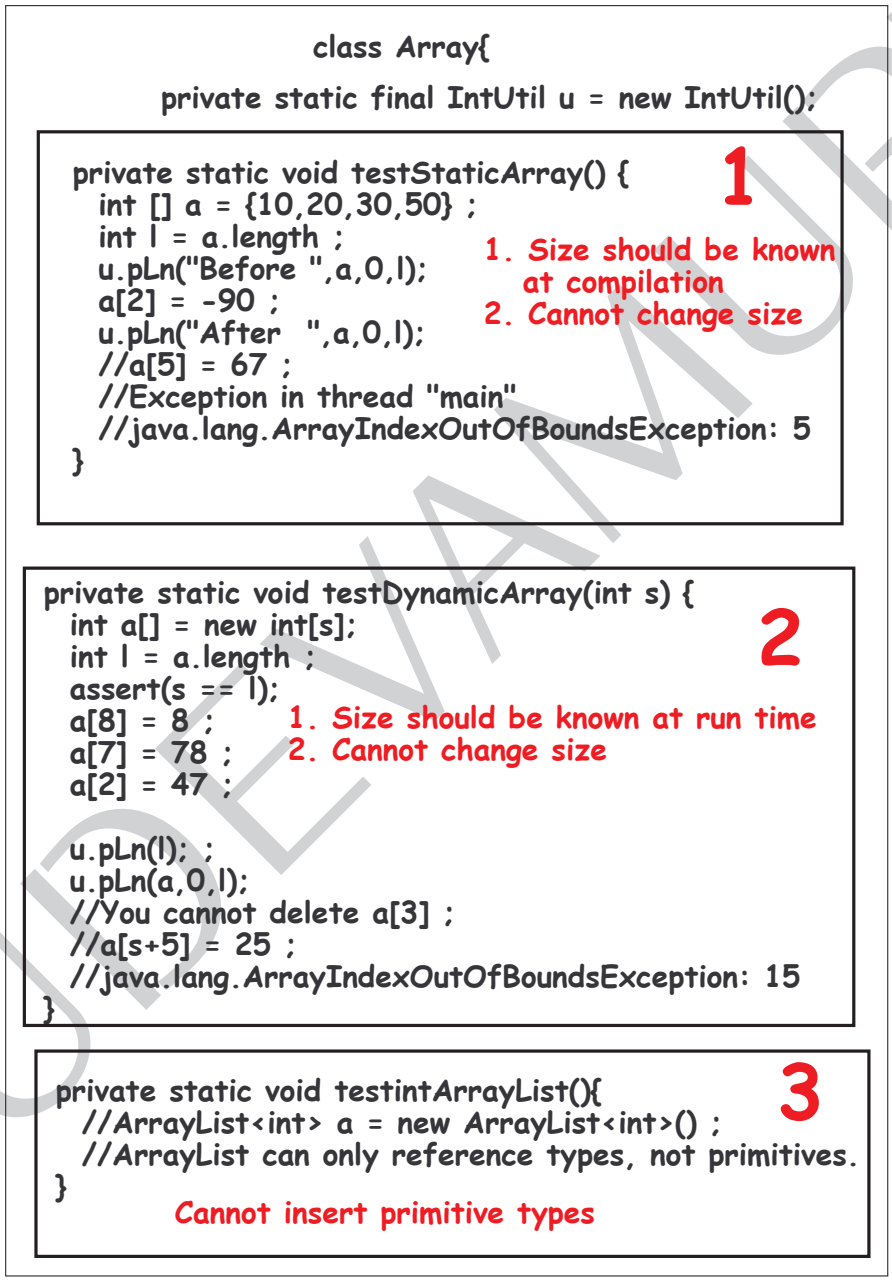

Figure 4.1: Various array's supported by Java

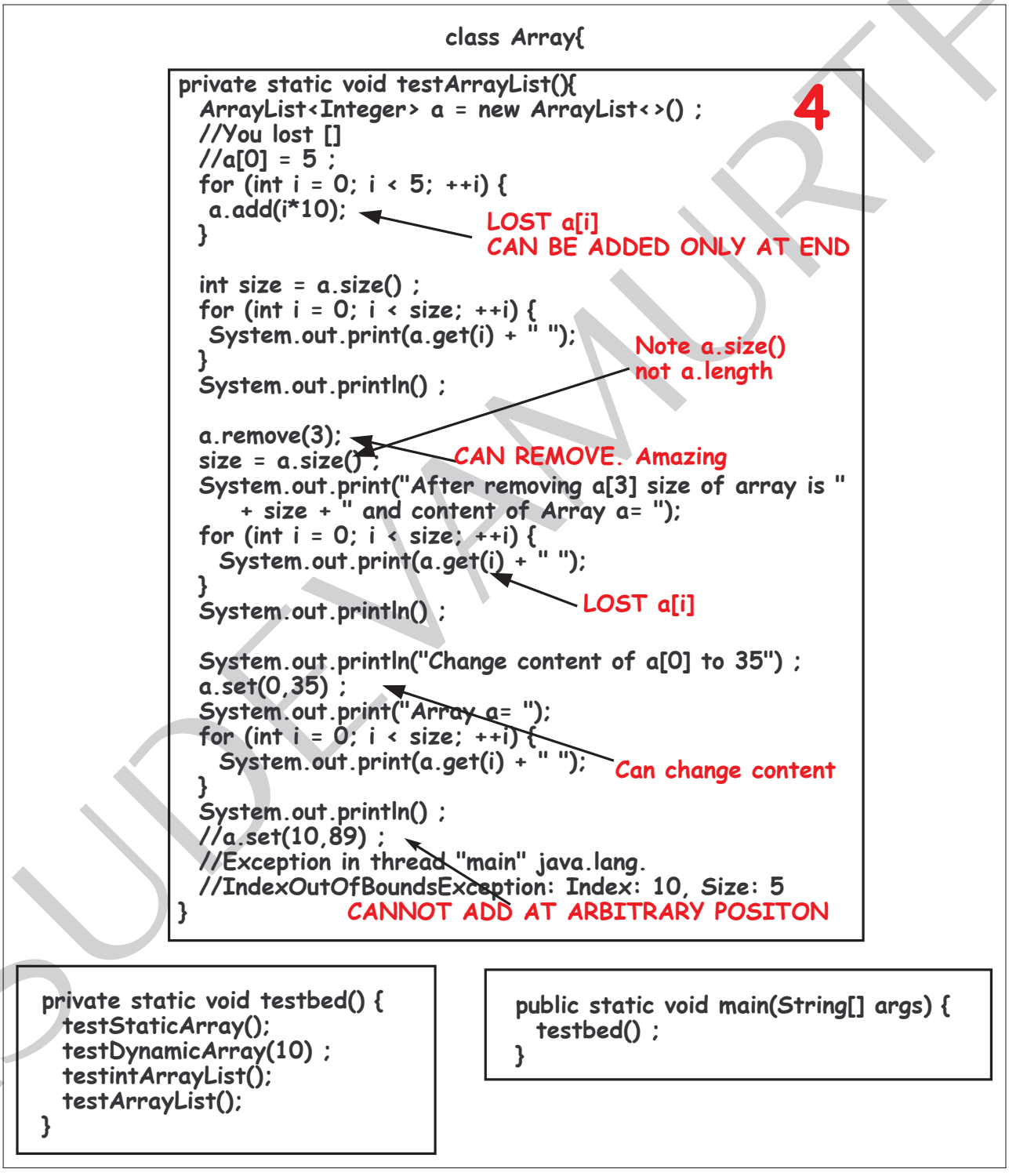

Figure 4.2: Various array's supported by Java

4.3 Dynamic growable int array

Figure 4.3: An **int** array that can grow

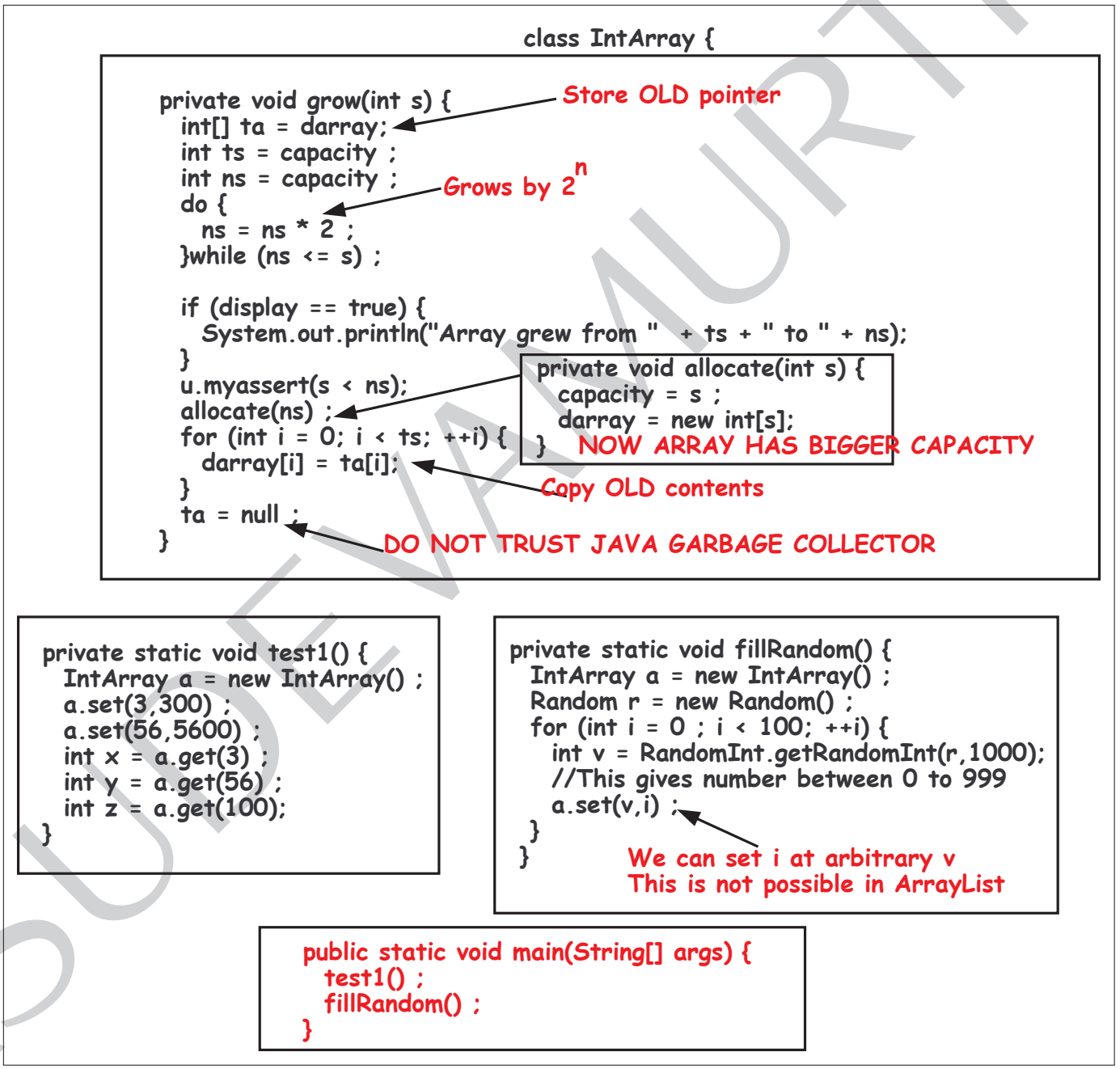

Figure 4.4: An **int** array that can grow

4.4 Dynamic growable char array

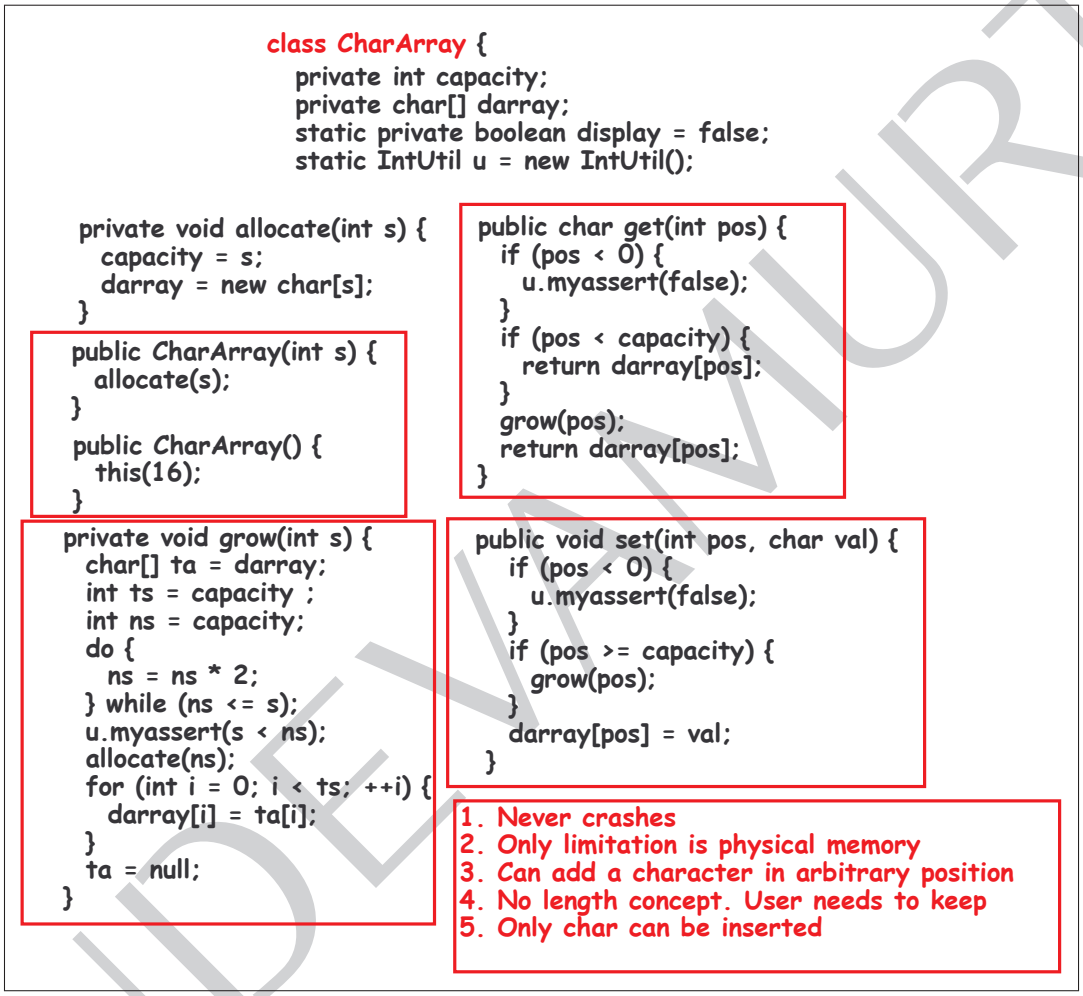

Figure 4.5: A **char** array that can grow

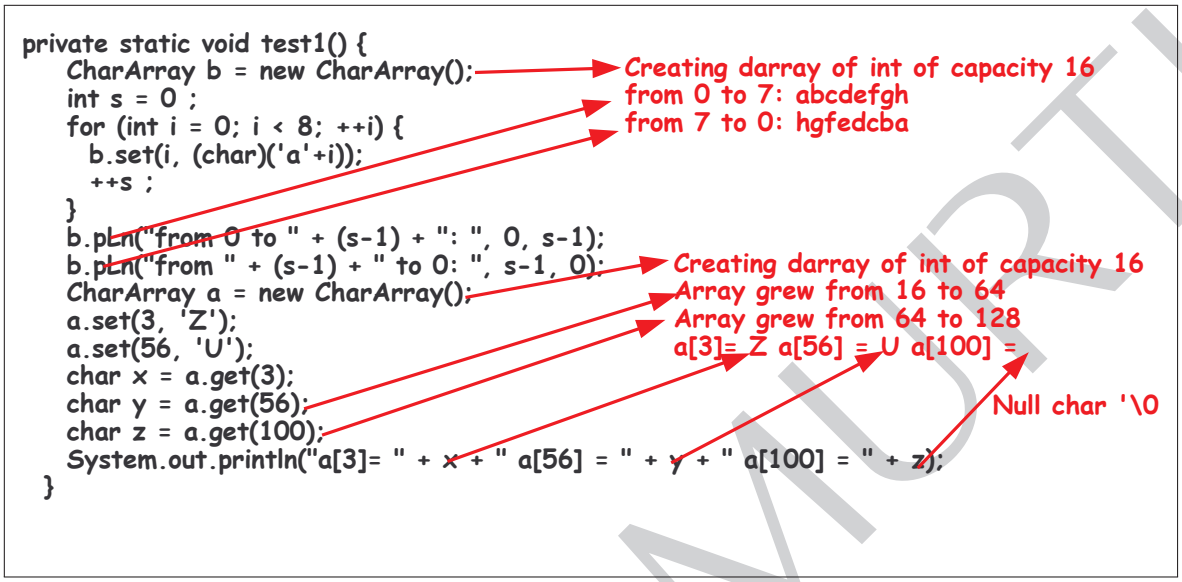

Figure 4.6: Using class *CharArray*

4.5 C string

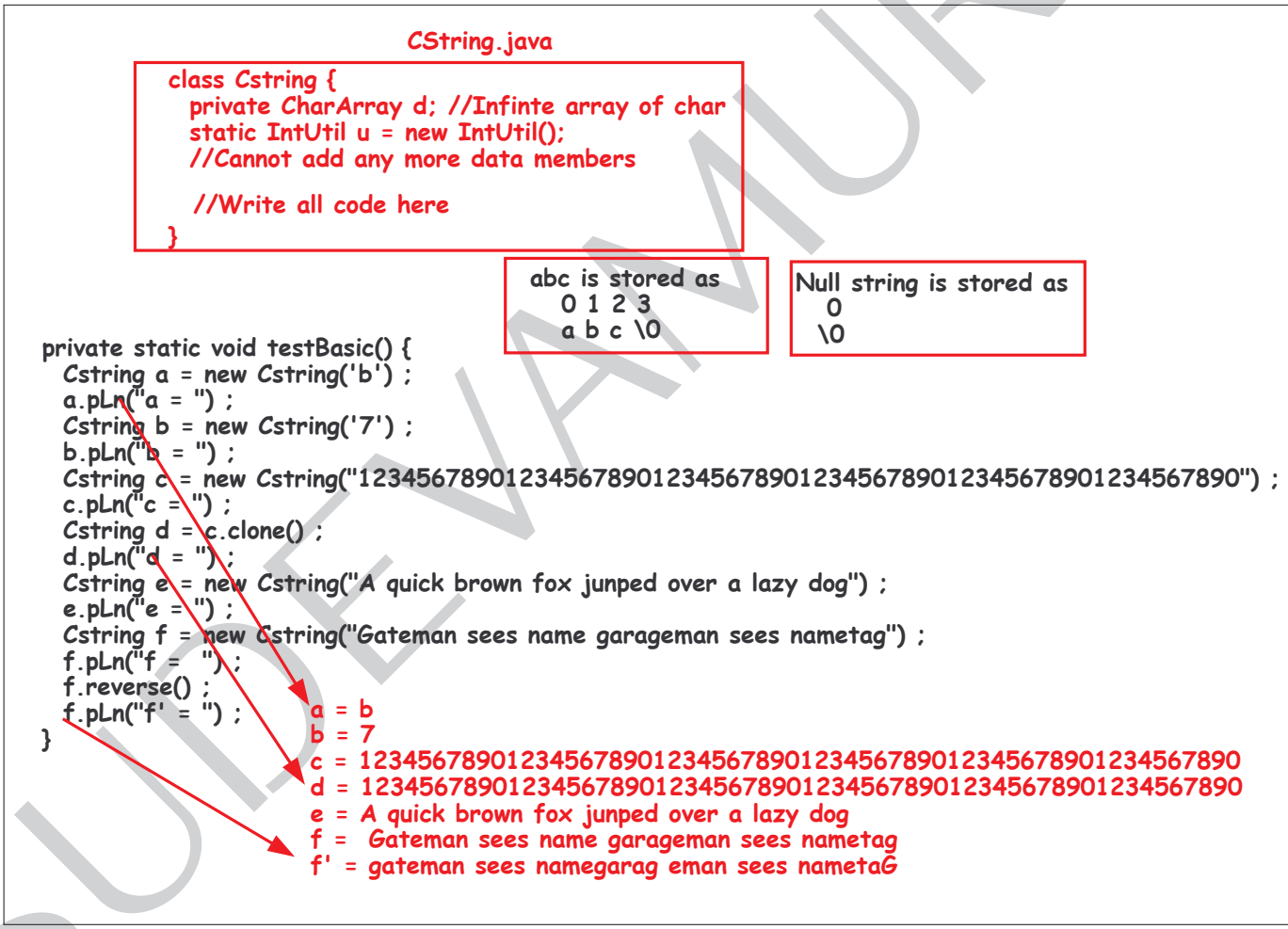

Figure 4.7: Mutable **CString** class

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("12345678901234567890123456789012345678901234567890123456789012345678901234567890") ;
   a.pLn("a = ") ;
   Cstring b = new Cstring("12345678901234567890123456789012345678901234567890123456789012345678901234567890") ;
   b.pLn("b = ") ;
   u.myassert(a.isEqual(b)) ;
   Cstring c = new Cstring("12345678901234567890123456789012345678901234567890123456789012345678901234567890") ;
   c.pLn("c = ") ;
   u.myassert(a.isEqual(c) == false) ;
}
```

write isEqual

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

4.6 Big Number

BigNumber.java

```
class BigNumber {
  private Cstring d; //data
  static IntUtil u = new IntUtil();

  //Cannot add any more data members
  //Cannot use ay java library function except printing
}
```

BigNumberTester.java

```
class BigNumberTester {
  static IntUtil u = new IntUtil();
}
private static void testBasic() {
  BigNumber a = new BigNumber(789) ;
  a.pLn("a = ") ;
  BigNumber b = new BigNumber('7') ;
  b.pLn("b = ") ;
  BigNumber c = new BigNumber("12345678901234567890123456789012345678901234567
  c.pLn("c = ") ;
  BigNumber d = c.clone() ;
  d.pLn("d = ") ;
}
```

```
a = 789
b = 7
c = 12345678901234567890123456789012345678901234567890123
d = 12345678901234567890123456789012345678901234567890123
```

Figure 4.9: **BigNumber** class and **BigNumberTester** class

4.6.1 Addition of big Number

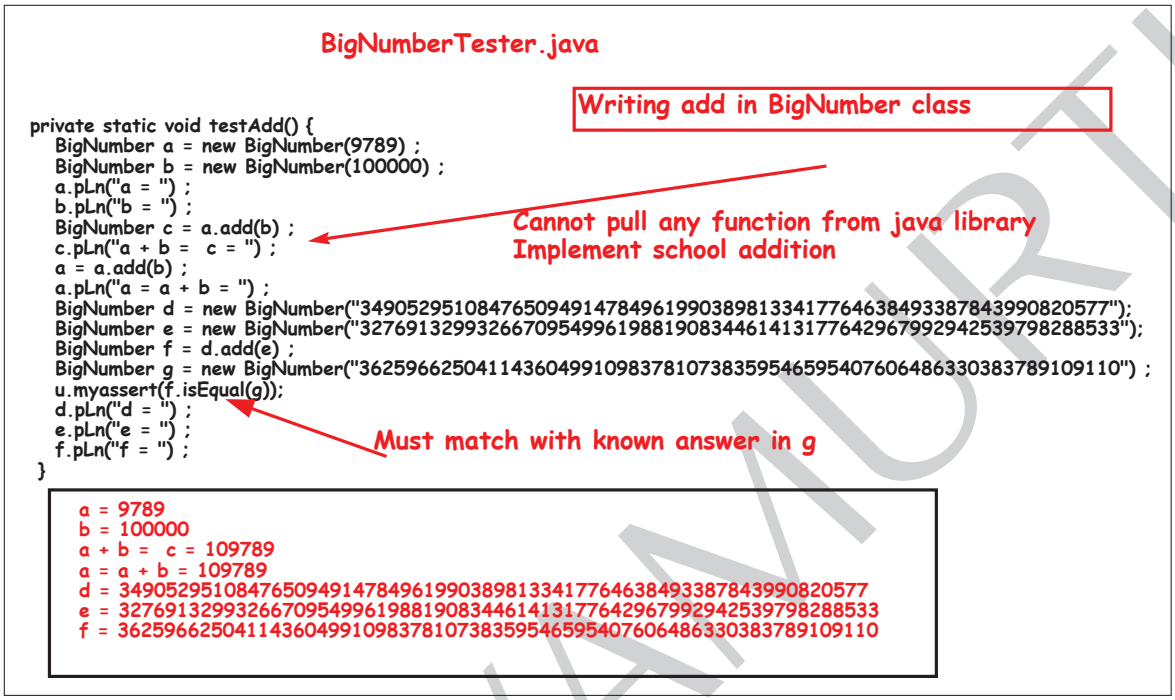

Figure 4.10: Writing *add* in **BigNumber** class

4.6.2 Subtraction of big Number

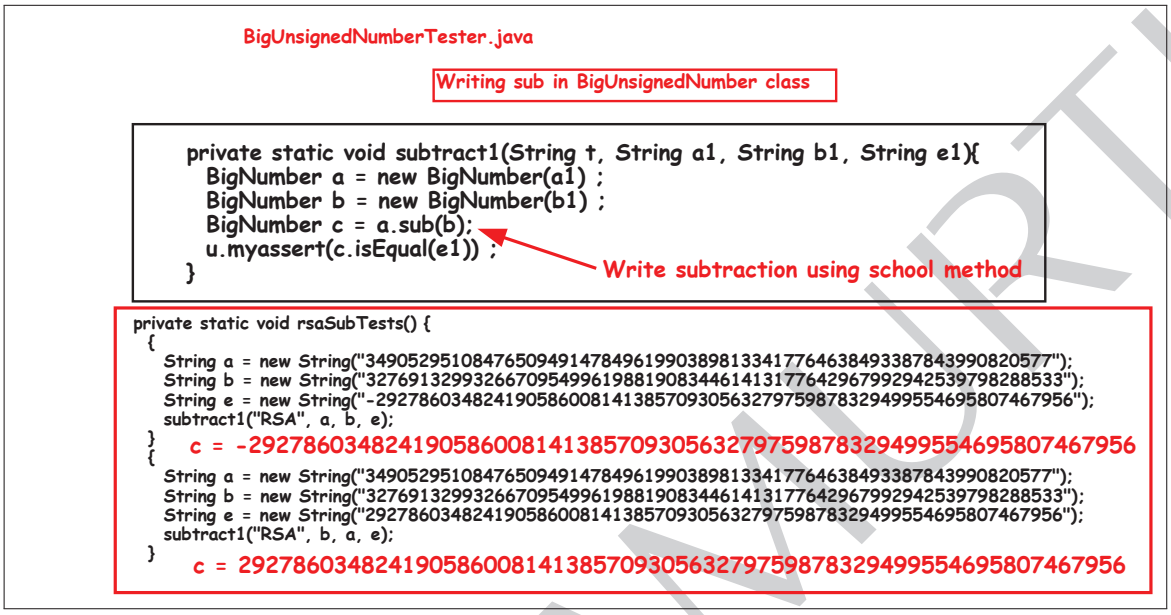

Figure 4.11: Writing *sub* in **BigNumber** class

4.6.3 Multiplication of big Number

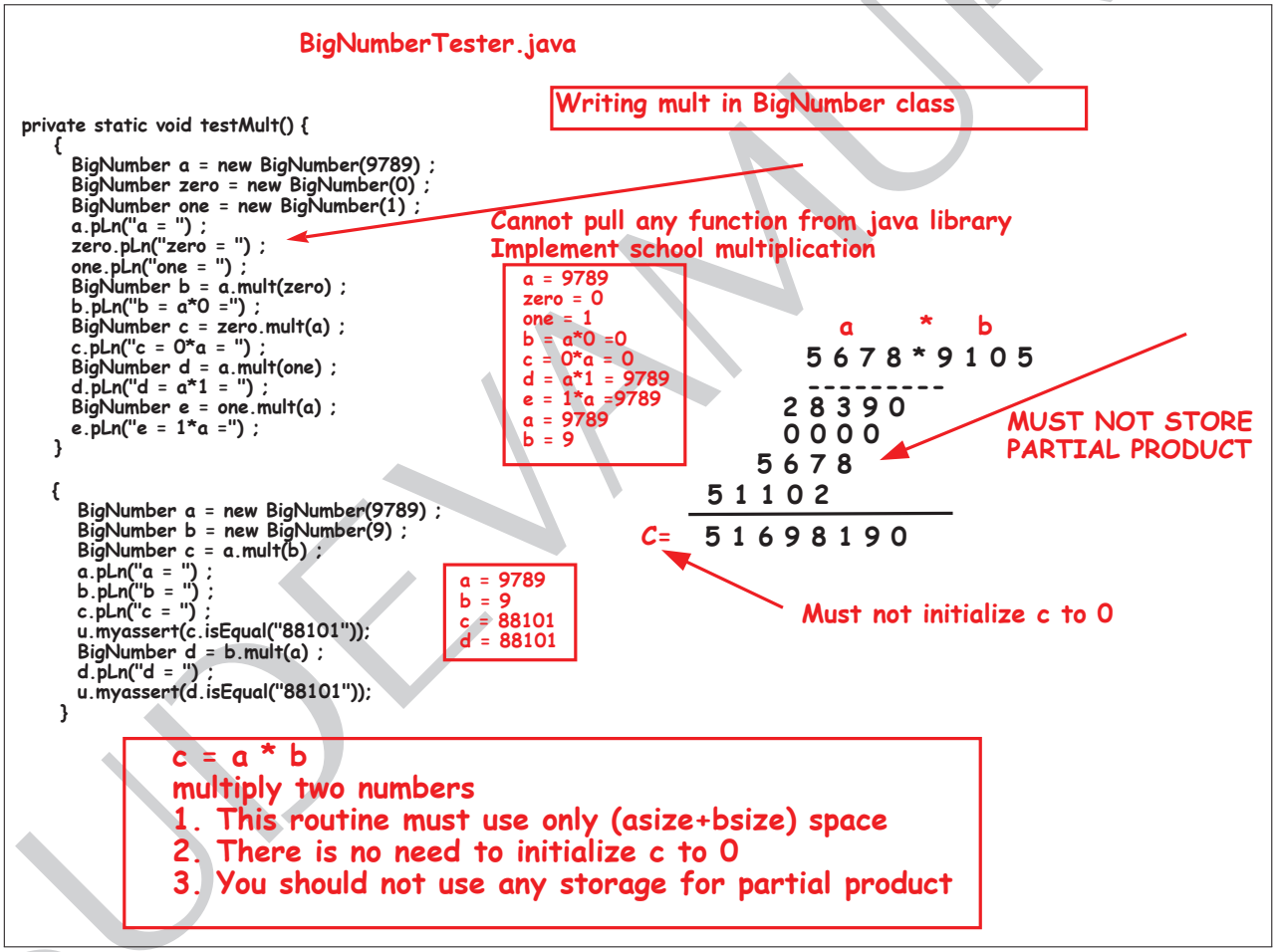

Figure 4.12: Writing *mult* in **BigNumber** class

BigNumberTester.java

Writing mult in BigNumber class

```
private static void testMult() {

 {
    BigNumber a = new BigNumber("34905295108476509491478496199038981334177646384933878439908205 77");
    BigNumber b = new BigNumber("32769132993266709549961988190834461413177642967992942539798288533");
    BigNumber c = new BigNumber("11438162575788886766923577997614661201021829672124236256256184293570693524573389783059712356395870505898907514759929002687954358429357069352457338978305971235639587050589890751475992900268795435 41");
    BigNumber d = b.mult(a) ;
    a.pLn("a = ") ;
    b.pLn("b = ") ;
    c.pLn("c = ") ;
    d.pLn("d = ") ;
    u.myassert(d.isEqual(c)) ;
 }
```

```
a = 34905295108476509491478496199038981334177646384933878439908205 77
b = 32769132993266709549961988190834461413177642967992942539798288533
c = 11438162575788886766923577997614661201021829672124236256256184293570693524573389783059712356395870505898907514759929002687954358429357069352457338978305971235639587050589890751475992900268795435 41
d = 11438162575788886766923577997614661201021829672124236256256184293570693524573389783059712356395870505898907514759929002687954358429357069352457338978305971235639587050589890751475992900268795435 41
```

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 pa

All asserts must pass

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

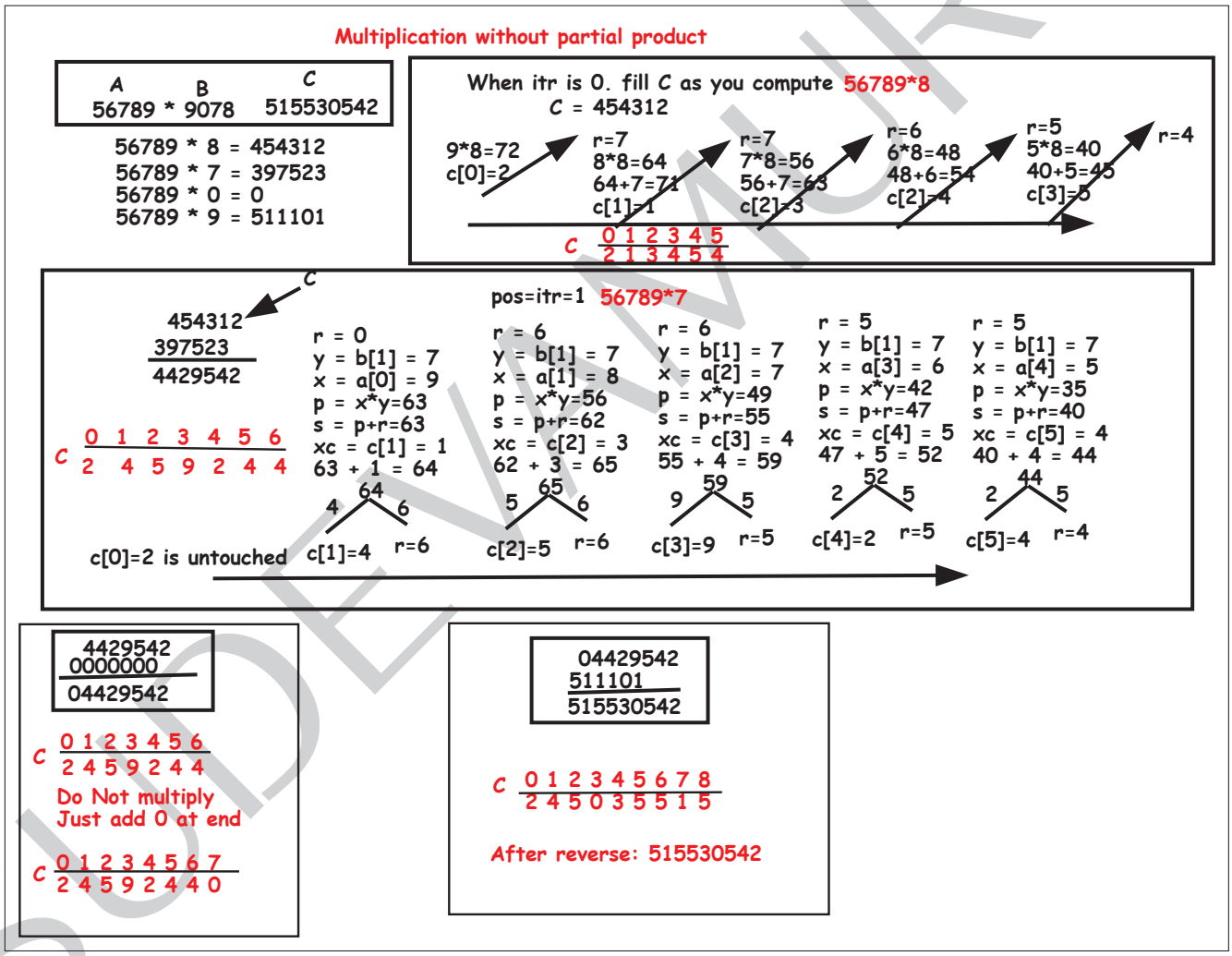

Figure 4.14: Multiplication without storing partial products

4.7 Computing factorial of a number

BigNumberTester.java

Writing factorial in BigNumber class

```
private static void testFact() {
    BigNumber fact100 = new BigNumber("933262 CUT FORSAVING PAPER 00000");
    BigNumber b = BigNumber.factorial(100) ;
    fact100.pLn("a = ") ;
    b.pLn("!100 = ") ;
    u.myassert(b.isEqual(fact100)) ;
    System.out.println("Number of digits in !100 = " + b.size()) ;

    BigNumber fact1000 = new BigNumber("40238 CUT FORSAVING PAPER 00000") ;
    long startTime = System.nanoTime();
    BigNumber c = BigNumber.factorial(1000) ;
    long endTime = System.nanoTime();
    double d = u.timeInSec(endTime,startTime) ;
    System.out.println("Run time for !1000 = " + d + " secs" );
    System.out.println("Number of digits in !1000 = " + c.size()) ;
    //Run time for !1000 = 0.225868438 secs
    //Number of digits in !1000 = 2568
    //fact1000.pLn("!1000 = ") ;
    //c.pLn("b = ") ;
    u.myassert(c.isEqual(fact1000)) ;
 }
```

```
a = "933262154439441526816992388562667004907159682643816214
685929638952175999932299156089414639761565182862536979
208272237582511852109168640000000000000000000000000"
```

```
Number of digits in !100 = 158
Run time for !1000 = 0.201742672 secs
Number of digits in !1000 = 2568
```

Figure 4.15: Writing *factorial* in **BigNumber** class


```
Run time for !1000 = 0.199138015 secs
Number of digits in !1000 = 2568
b = 402387260077093773543702433923003985719374864210714632543799
910429938512398629020592044208486969404800479988610197196058 6
316668729948085589013238296699445909974245040870737599188236 2
772718873251977950595099527612087497546249704360141827809464 6
496291056393887437886487337119181045825783647849977012476632 8
898359557354325131853239584630755574091142624174743493475534 2
864657661166779739666882029120737914385371958824980812686783 8
374559731746136085379534524221586593201928090878297308431392 8
444032812315586110369768013573042161687476096758713483120254 7
858932076716913244842623613141250878020800026168315102734182 7
977704784635868170164365024153691398281264810213092761244896 3
599287051149649754199093422215668325720808213331861168115536 1
583654698404670897560290095053761647584772842188967964624494 5
160765353408198901385442487984959953319101723355556602139450 3
997362807501378376153071277619268490343526252000158885351473 3
161170210396817592151090778801939317811419454525722386554146 1
062892187960223838971476088506276862967146674697562911234082 4
392081601537808898939645182632436716167621791689097799119037 5
403127462228998800519544441428201218736174599264295658174662 8
302955570299024324153181617210465832036786906117260158783520 7
515162842255402651704833042261439742869330616908979684825901 2
545832716822645806652676995865268227280707578139185817888965 2
208164348344825993266043367660176999612831860788386150279465 9
551311565520360939881806121385586003014356945272242063446317 9
746059468257310379008402443243846565724501440282188525247093 5
190620929023136493273497565513958720559654228749774011413346 9
627154228458623773875382304838656889764619273838149001407673 1
044664025989949022222176590433990188601856652648506179970235 6
193897017860040811889729918311021171229845901641921068884387 1
218556461249607987229085192968193723886426148396573822911231 2
502418664935314397013742853192664987533721894069428143411852 0
158014123344828015051399694290153483077644569099073152433278 2
882698646027898643211390835062170950025973898635542771967428 2
224875758676575234422020757363056949882508796892816275384886 3
396909959826280956121450994871701244516461260379029309120889 0
869420285106401821543994571568059418727489980942547421735824 0
106367740459574178516082923013535808184009699637252423056085 5
903700624271243416909004153690105933983835777939410970027753 4
720000000000000000000000000000000000000000000000000000000000 0
000000000000000000000000000000000000000000000000000000000000 0
000000000000000000000000000000000000000000000000000000000000 0
000000000000000000000000000000000000000000000000000000000000 0
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

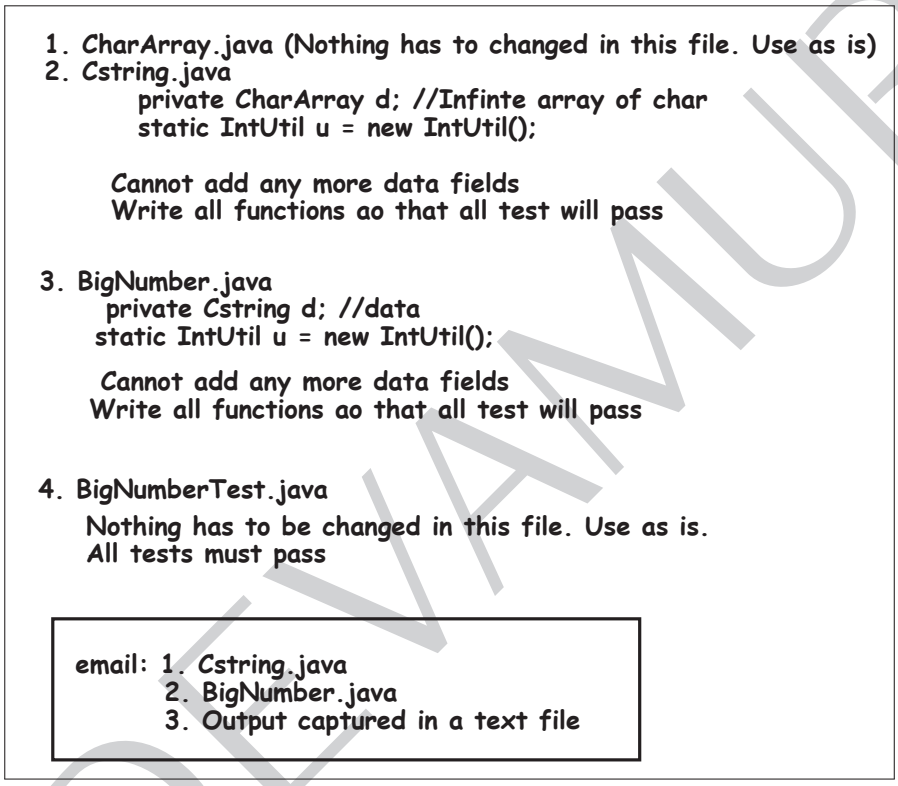

Figure 4.17: Implementing *Cstring* class and *BigNumber* class

