Chapter 1

Overview

CSC 113
King Saud University
College of Computer and Information Sciences
Department of Computer Science

Dr. S. HAMMAMI

Objectives

- After you have read and studied this chapter, you should be able to
    - Define a class with multiple methods and data members
    - Define and use value-returning methods
    - Pass both primitive data and objects to a method
    - Manipulate a collection of data values, using an array.
    - Declare and use an array of objects in writing a program

OUTLINE

1. Passing Objects to a Method
2. Returning an Object From a Method
3. The Use of this in the add Method
4. Overloaded Methods
5. Arrays of Objects
6. Examples

1. Passing Objects to a Method

- As we can pass **int** and **double** values, we can also pass an **object** to a method.

- When we pass an object, we are actually passing the **reference** (name) of an object
  - it means a duplicate of an object is NOT created in the called method

LibraryCard class: A LibraryCard object is owned by a Student, and it records the number of books being checked out.

Student class

```java
/*
   File: Student.java
*/

class Student {
   // Data Member
   private String name;
   private String email;

   //Constructor
   public Student() {
      name = "Unassigned";
      email = "Unassigned";
   }
   //Returns the email of this student
   public String getEmail( ) {
      return email;
   }
   //Returns the name of this student
   public String getName( ) {
           return name;
   }
   //Assigns the name of this student
   public void setName(String studentName) {
      name = studentName;
   }
   //Assigns the email of this student
   public void setEmail(String address) {
      email = address;
   }
 }
```

LibraryCard class

```java
/*
   File: LibraryCard.java
 */
class LibraryCard {
   //student owner of this card
   private Student owner;
   //number of books borrowed
   private int borrowCnt;
   //numOfBooks are checked out
   public void checkOut(int numOfBooks) {
      borrowCnt = borrowCnt + numOfBooks;
   }
   //Returns the name of the owner of this card
   public String getOwnerName( ) {
      return owner.getName( );
   }
   //Returns the number of books borrowed
   public int getNumberOfBooks( ) {
      return borrowCnt;
   }
   //Sets the owner of this card to student
   public void setOwner(Student student) {
      owner = student;
   }
   //Returns the string representation of this card
   public String display( ) {
      return  "Owner Name:    " + owner.getName( ) + "\n" +
            "      Email:    " + owner.getEmail( ) + "\n" +
            "Books Borrowed: " + borrowCnt;
   }
 }
```

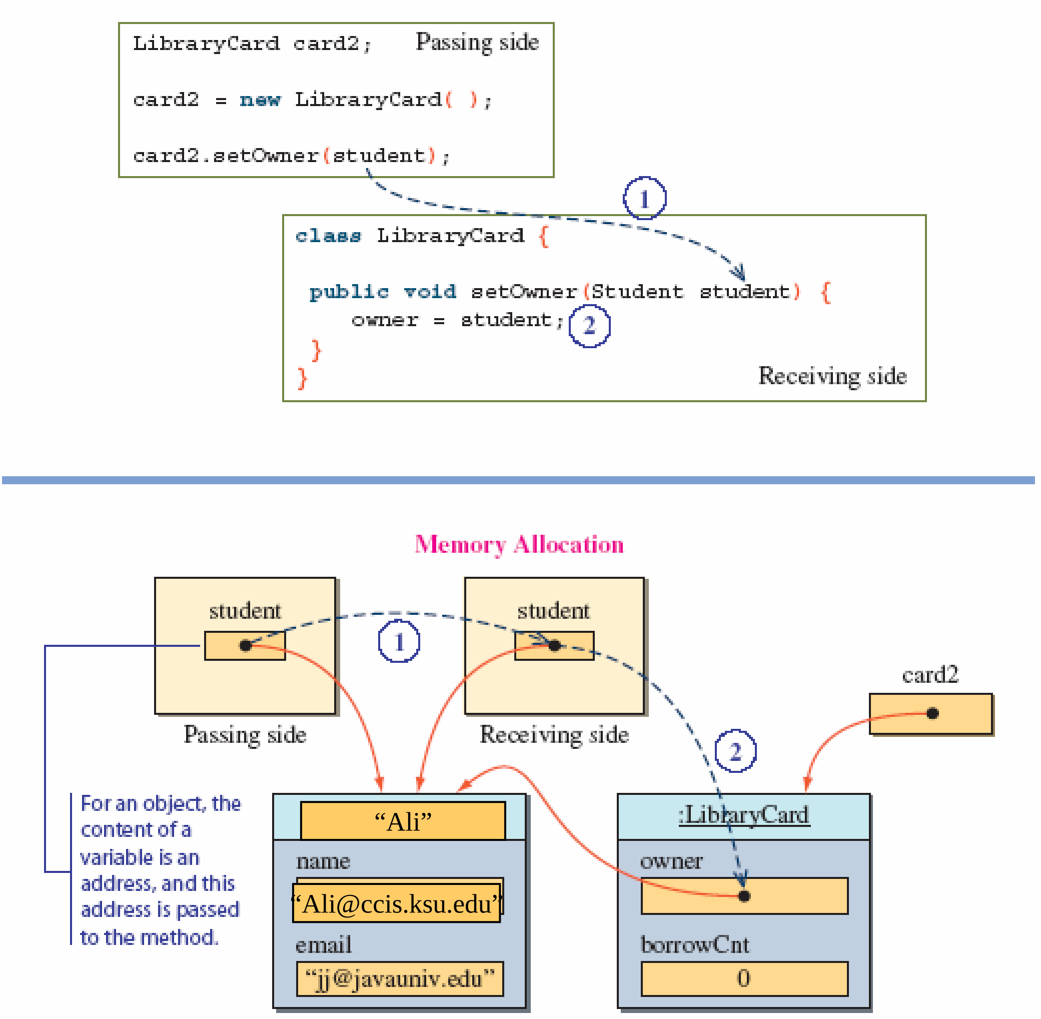
Passing a Student Object

Suppose a single student owns two library cards. Then we can make the data member owner of two LibraryCard Objects to refer to the same Student object. Here’s one such program

```
/*
   File: Librarian.java
*/
class Librarian {
   public static void main( String[] args ) {

      Student    student;
      LibraryCard card1, card2;

      student = new Student( );
      student.setName(“Ali");
      student.setEmail(“ali@ccis.ksu.edu");

      card1 = new LibraryCard( );
      card1.setOwner(student);
      card1.checkOut(3);

      card2 = new LibraryCard( );
      card2.setOwner(student); //the same student is the owner
                                   //of the second card, too

      System.out.println("Card1 Info:");
      System.out.println(card1.display() + "\n");

      System.out.println("Card2 Info:");
      System.out.println(card2.display() + "\n");

   }
}
```

Passing a Student Object

When we say pass an object to a method, we are not sending a copy of an object, but rather a reference to the object.
This diagram illustrates how an objects is passed as an arguments to a method



Passing side

```
LibraryCard card2;

card2 = new LibraryCard( );

card2.setOwner(student);
```

Receiving side

```
class LibraryCard {

 public void setOwner(Student student) {
    owner = student;
 }
}
```

For an object, the content of a variable is an address, and this address is passed to the method.

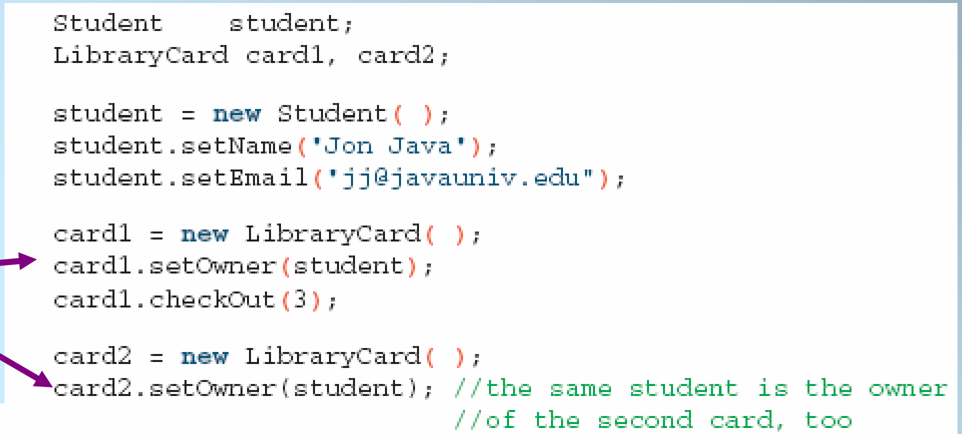
Sharing an Object

In this program, we create one Student object. Then we create two LibraryCard objects. For each of these LibraryCard objects, we pass the same student when calling their setOwner methods:

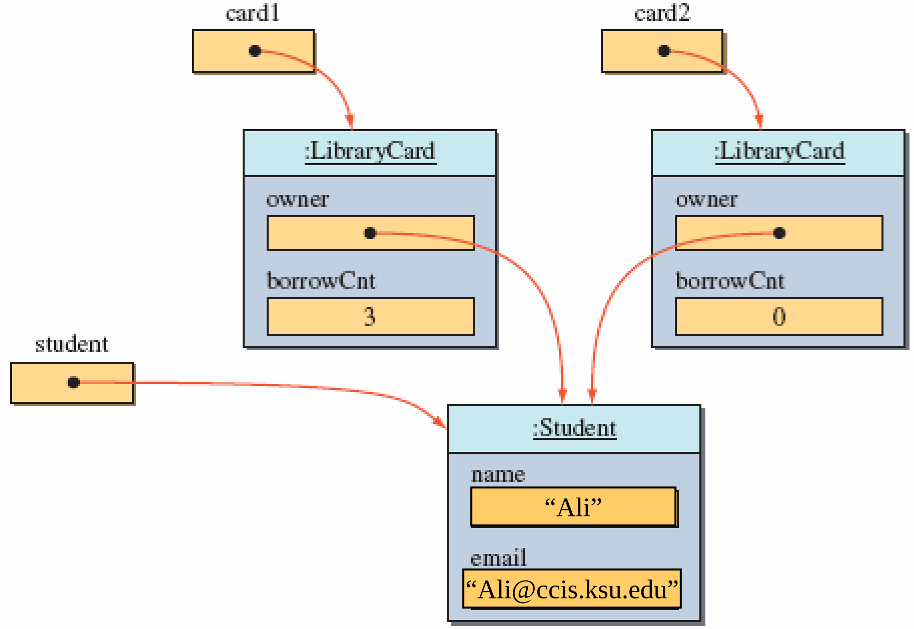
card1.setOwner(student);
….
card2.setOwner(student);

After the setOwner method of card2 is called in the main method, we have the following state of memory.

- We pass the same Student object to card1 and card2



```
Student     student;
LibraryCard card1, card2;

student = new Student( );
student.setName("Jon Java");
student.setEmail("jj@javauniv.edu");

card1 = new LibraryCard( );
card1.setOwner(student);
card1.checkOut(3);

card2 = new LibraryCard( );
card2.setOwner(student); //the same student is the owner
                         //of the second card, too
```



- Since we are actually passing a reference to the same object, it results in the owner of two LibraryCard objects pointing to the same Student object

2. Returning an Object From a Method

- As we can return a primitive data value from a method, we can return an object from a method also.
- We return an object from a method, we are actually returning a reference (or an address) of an object.
  - This means we are not returning a copy of an object, but only the reference of this object

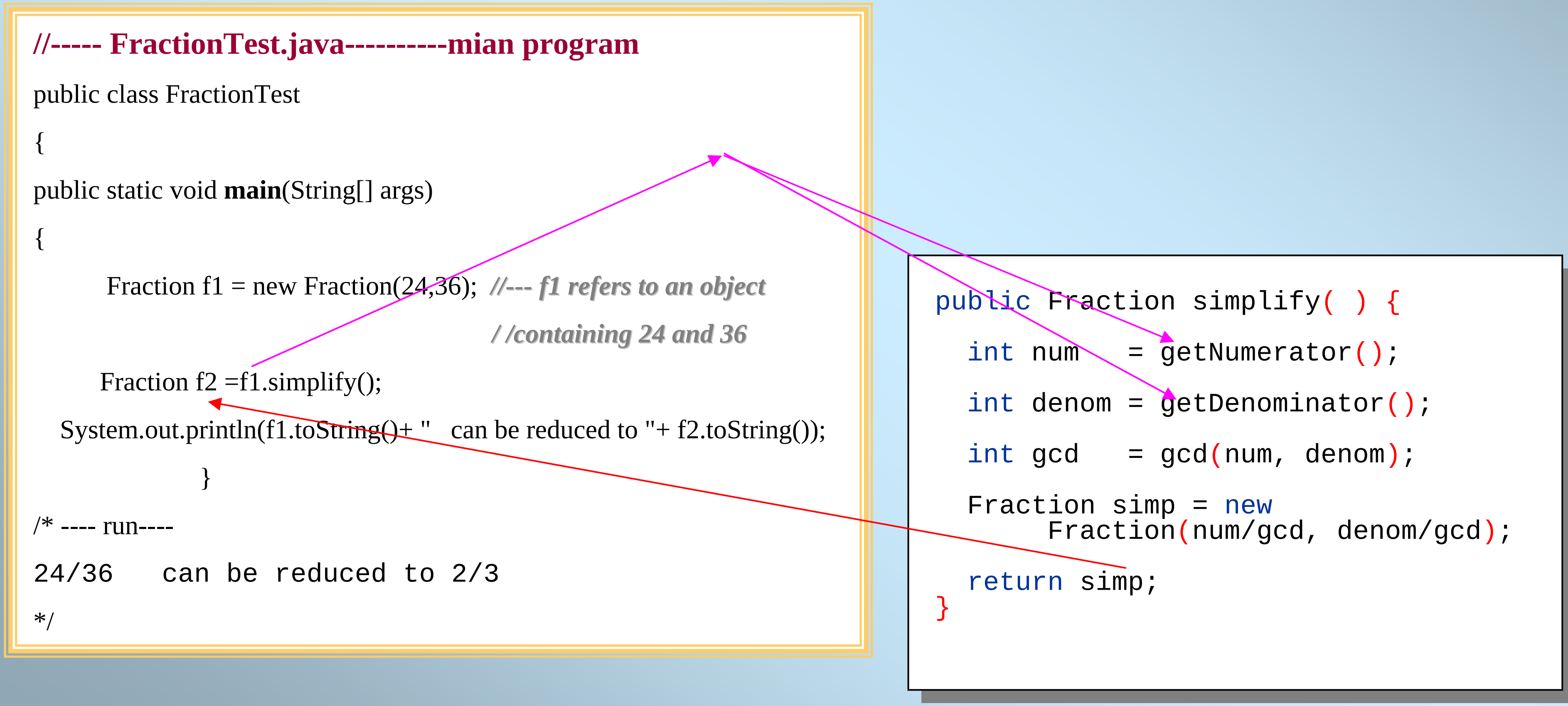
```
//== Class Fraction============
public class Fraction
{  private int numerator;
    private int  denominator;
    //===== Constructors ========//
    public Fraction()     {this(0,1);  }
    public Fraction(int number) {this(number,1); }
   public Fraction(Fraction frac)
   {this(frac.getNumerator(), frac.getDenominator()) }
    public Fraction(int num, int denom)
  {setNumerator(num); setDenominator(denom); }
  //======Public Instance Methods =============
   public int getNumerator()  {return (numerator); }
  public int getDenominator() { return (denominator); }
  public void setNumerator(int num) {numerator=num; }
  public void setDenominator(int denom)
 {   if (denom == 0)
      {  System.out.println("Fatal error, divid by zero");
       System.exit(1);
      }
    denominator=denom;
 }
```

```
//== Class Fraction:  continue ============
//---- sum = this + frac ---------
public Fraction add(Fraction frac)
{   int n1,d1, n2,d2;
    n1=this.getNumerator();  d1=this.getDenominator();
    n2=frac.getNumerator();  d2=frac.getDenominator();
    Fraction sum =new Fraction(n1*d2+d1*n2, d1*d2);
    return(sum);
}
//----- sum = this + number ---------
public Fraction add(int number)
{  Fraction frac = new Fraction(number, 1);
   Fraction sum = this.add(frac);
   return(sum);
}
//----- sub = this - frac ----------
public Fraction subtract(Fraction frac)
{  int n1,d1, n2,d2;
   n1=numerator;      d1=denominator;
   n2=frac.numerator;  d2=frac.denominator;
    Fraction sub =new Fraction(n1*d2-d1*n2, d1*d2);
   return(sub);
}
//----- sub = this - number ---------
public Fraction subtract(int number)
{  Fraction frac = new Fraction(number, 1);
    return(this.subtract(frac));
}
```

```
//== Class Fraction continue ==========
//---- mult = this * frac ---------
public Fraction multiply(Fraction frac)
{
    int n1,d1, n2,d2;
    n1=this.getNumerator(); d1=this.getDenominator();
    n2=frac.getNumerator();  d2=frac.getDenominator();
    Fraction mult =new Fraction(n1*n2, d1*d2);
     return(mult);
 }
   //----- mult = this * number ---------
  public Fraction multiply(int number)
 {
    Fraction frac = new Fraction(number, 1);
    return(this.multiply(frac));
}
//---- div = this / frac ---------
public Fraction divide(Fraction frac)
{
    int n1,d1, n2,d2;   n1=numerator;   d1=denominator;
    n2=frac.getNumerator();    d2=frac.getDenominator();
   Fraction div =new Fraction(n1*d2, d1*n2);return(div);
}
 //----- mult = this / number ---------
  public Fraction devide(int number)
   {
      Fraction frac = new Fraction(number, 1);
      return(this.divide(frac));   }
   public boolean equals(Fraction frac)
    {
 Fraction f1 =this.simplify();
Fraction f2 =frac.simplify();
if ((f1.getDenominator()== f2.getDenominator()) &&
f1.getNumerator()== f2.getNumerator())  return true;
return false;
}
```

```
//== Class Fraction:  continue ============
public Fraction simplify()
{
      int num =getNumerator();     int denom= this.getDenominator();
     int gcd =this.gcd(num,denom );
    Fraction simp =new Fraction(num/gcd, denom/gcd);
     return(simp);
}
public String toString()
{
return (this.getNumerator() + "/" + this.getDenominator());
}
//====Class Methods================
public static int gcd(int m,int n)
{
     int r= n%m;
     while(r !=0) { n=m;    m=r;  r=n%m;}  return (m);
}
public static Fraction minimum(Fraction f1, Fraction f2)
{
     double  dec1 = f1.decimal();
     double  dec2 = f2.decimal();
     if (dec1 < dec2)  return (f1);
         return f2;
}
//======= Private Methods==============
private double decimal()
{
  return(this.getNumerator()/ this.getDenominator());
}

}//---- end of calss Fraction---
```

When we say “return an object from a method”, we are actually returning the address, or the reference, of an object to the caller

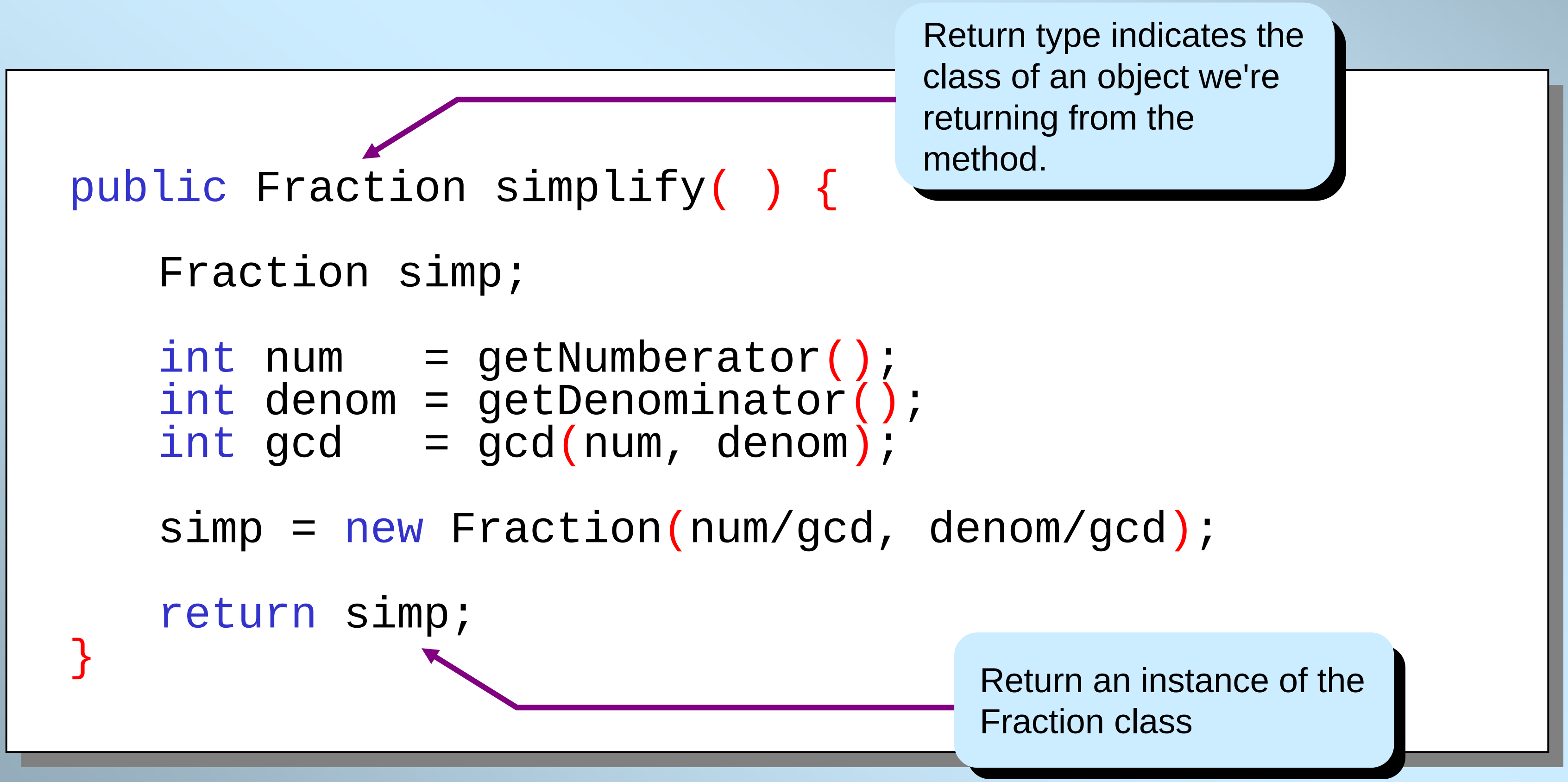


```
//----- FractionTest.java----------mian program
public class FractionTest
{
public static void main(String[] args)
{
        Fraction f1 = new Fraction(24,36);  //--- f1 refers to an object
                                            / /containing 24 and 36
       Fraction f2 =f1.simplify();
   System.out.println(f1.toString()+ "   can be reduced to "+ f2.toString());
                }
/* ---- run----
24/36   can be reduced to 2/3
*/
```

```
public Fraction simplify( ) {
   int num   = getNumerator();
   int denom = getDenominator();
   int gcd   = gcd(num, denom);
   Fraction simp = new
        Fraction(num/gcd, denom/gcd);
   return simp;
}
```

Sample Object-Returning Method

- Here's a sample method that returns an object:



```
public Fraction simplify( ) {

    Fraction simp;

    int num   = getNumberator();
    int denom = getDenominator();
    int gcd   = gcd(num, denom);

    simp = new Fraction(num/gcd, denom/gcd);

    return simp;
}
```

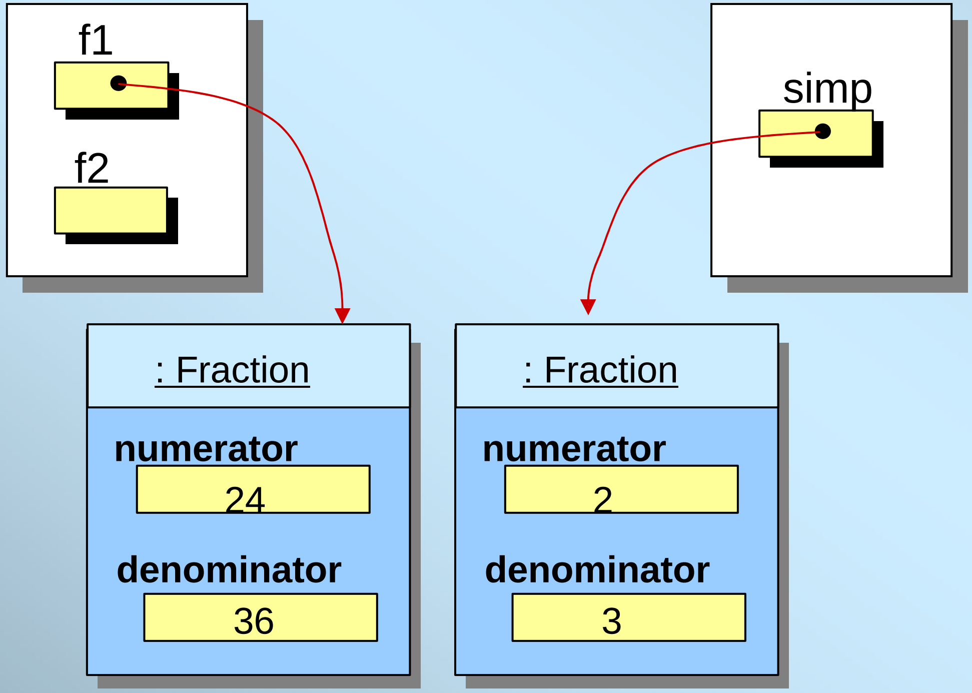
Return type indicates the class of an object we're returning from the method.

Return an instance of the Fraction class

A Sample Call to simplify

```
f1 = new Fraction(24, 26);
```

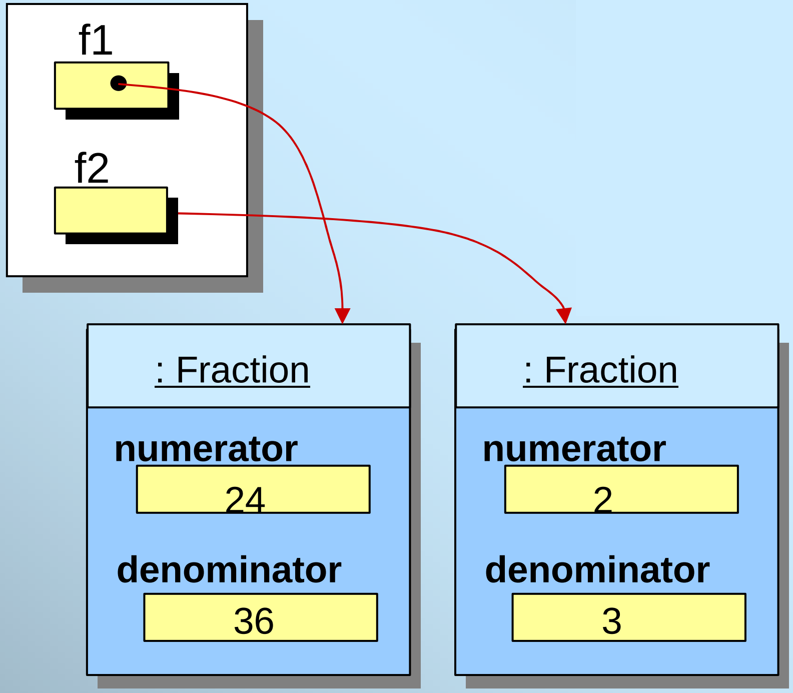
```
public Fraction simplify( ) {

   int num   = getNumerator();
   int denom = getDenominator();
   int gcd   = gcd(num, denom);

   Fraction simp = new
        Fraction(num/gcd, denom/gcd);

   return simp;
}
```



A Sample Call to simplify (cont'd)

```
f1 = new Fraction(24, 26);
```

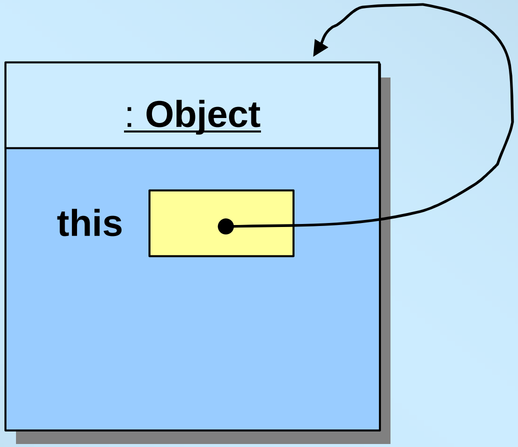
```
public Fraction simplify( ) {

   int num   = getNumerator();
   int denom = getDenominator();
   int gcd   = gcd(num, denom);

   Fraction simp = new
        Fraction(num/gcd, denom/gcd);

   return simp;
}
```



The value of simp, which is a reference, is returned and assigned to f2.

Reserved Word this

- The reserved word this is called a *self-referencing pointer* because *it refers to an object* from the object's method.

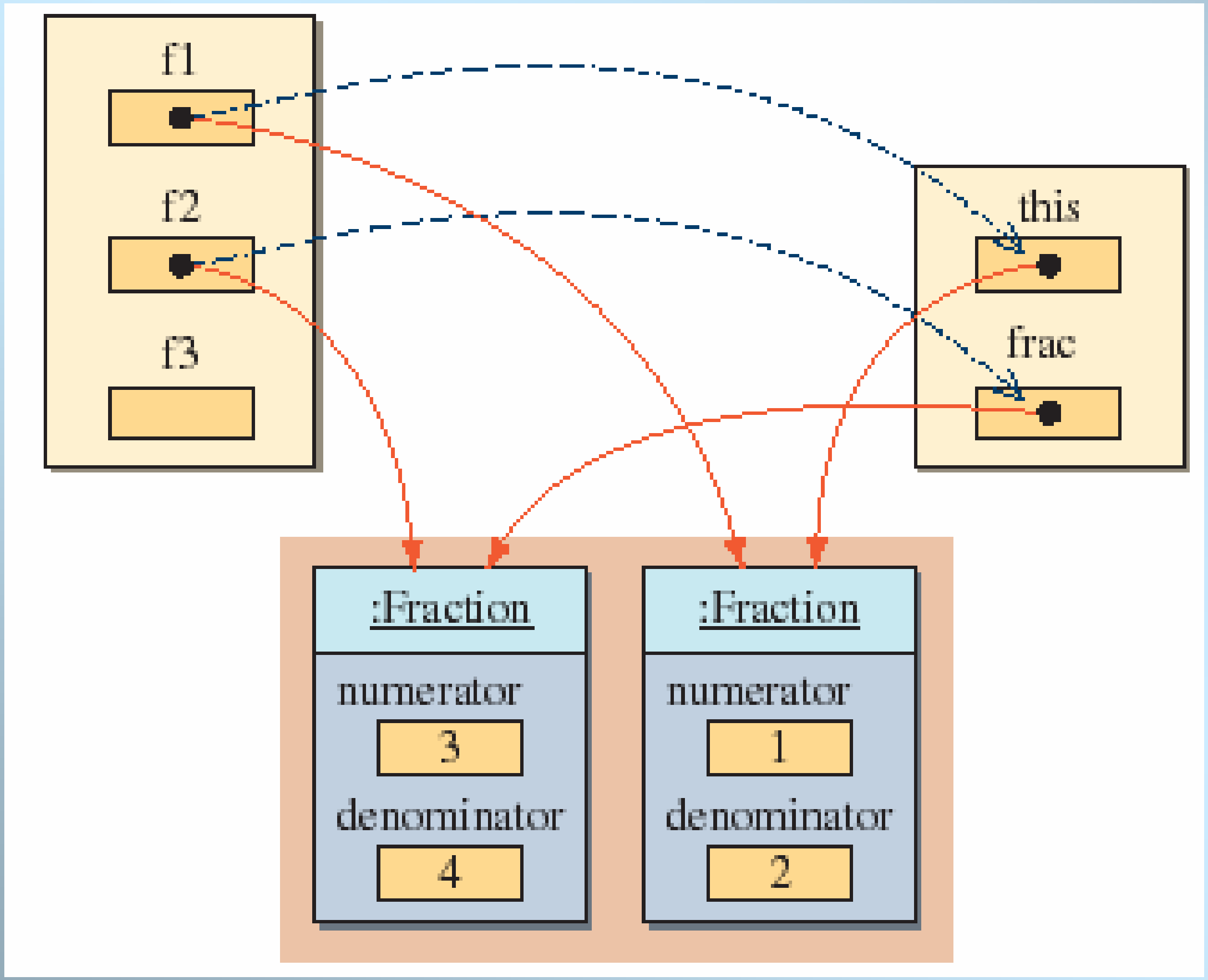


- The reserved word this can be used in different ways. We will see all uses in this chapter.

3. The Use of this in the add Method

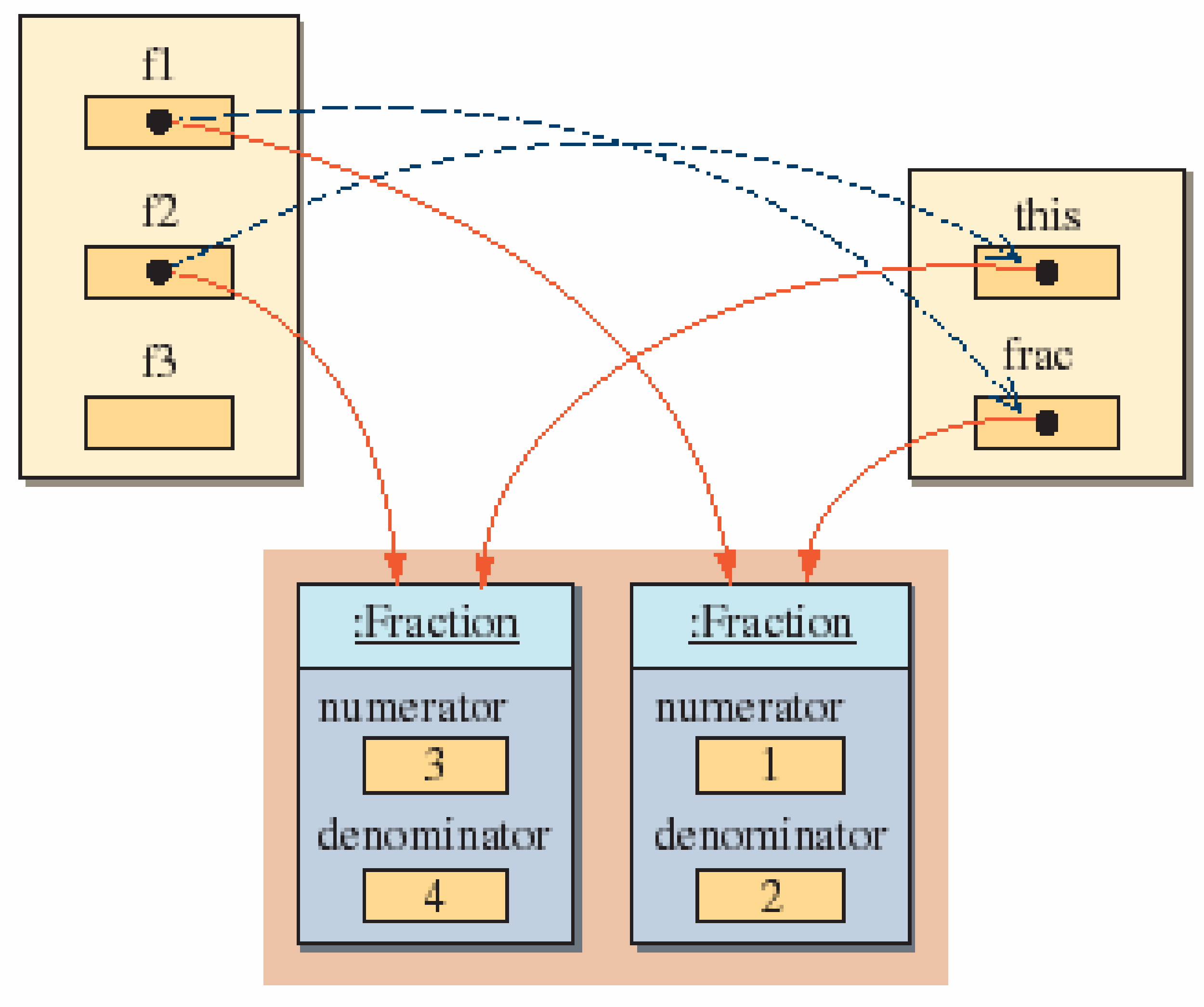
```
public Fraction add(Fraction frac) {

   int      a, b, c, d;
   Fraction sum;

   a = this.getNumerator();   //get the receiving
   b = this.getDenominator(); //object's num and denom

   c = frac.getNumerator();   //get frac's num
   d = frac.getDenominator(); //and denom

   sum = new Fraction(a*d + b*c, b*d);

   return sum;
}
```

f3 = f1.add(f2)



Because f1 is the receiving object (we're calling f1's method), so the reserved word this is referring to f1.

f3 = f2.add(f1)



This time, we're calling f2's method, so the reserved word this is referring to f2.

Using this to Refer to Data Members

- In the previous example, we showed the use of this to call a method of a receiving object.
- It can be used to refer to a data member as well.

```
class Person {

   int  age;

    public void setAge(int val) {
       this.age = val;
   }
   . . .
}
```

4. Overloaded Methods

- Methods can share the same name as long as
  - they have a different number of parameters (Rule 1) or
  - their parameters are of different data types when the number of parameters is the same (Rule 2)

Note: It is not necessary to create an object for f3 and f4

```
//----- FractionTest.java----------mian program

public class FractionTest {

public static void main(String[] args)   {

         Fraction f1, f2, f3,f4;

         f1 = new Fraction(3,4);  //-- create an object for f1

         f2 = new Fraction(2,5);  //--create and object for f2

         f3=f1.multiply(f2);   //--- f3 = f1 x f2  = 6 / 20

          f4=f1.multiply(6);   //--- f4 = f1 x 6   = 18 / 4

         System.out.println(" f3 =  "+ f3.toString()+

                              "  and f4 =  "+ f4.toString());
}

/* ---- run----

 f3 =  6/20  and f4 =  18/4

*/
```

```
//---- mult = this * frac ---------
public Fraction multiply(Fraction frac)
{
   int n1,d1, n2,d2;
   n1=this.getNumerator(); d1=this.getDenominator();
   n2=frac.getNumerator();  d2=frac.getDenominator();
   Fraction mult =new Fraction(n1*n2, d1*d2);
    return(mult);
 }
  //----- mult = this * number ---------
 public Fraction multiply(int number)
 {
   Fraction frac = new Fraction(number, 1);
   return(this.multiply(frac));
}
```

5. Arrays of Objects

- In Java, in addition to arrays of primitive data types, we can declare arrays of objects

- An array of primitive data is a powerful tool, but an array of objects is even more powerful.

- The use of an array of objects allows us to model the application more cleanly and logically.

The Person Class

- We will use Student objects to illustrate the use of an array of objects.

```
public class Person
{
            private String name;
            private int age;
            private char gender;
            public Person()    {age=0; name=" "; gender=' ';}
            public Person(String na, int ag, char gen)  {setAge(ag); setName(na); setGender(gen); }
            public Person(Person pr)       { setPerson(pr);}
            public void setPerson(Person p)
            { age=p.age; gender =p.gender;
             name=p.name. substring(0, p.name.length());      }
            public void setAge (int a) {age=a;}
            public void setGender (char g) {gender=g;}
            public void setName(String na)
             {name= new String(na);}
            public int getAge(){return age;}
            public char getGender () {return gender;}
            public String getName () { return name;}
}
```

Creating an Object Array - 1

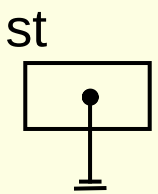
Code

A

```
Student[ ]  st;

st = new Student[20];

st[0] = new Student( );
```

Only the name pr is declared, no array is allocated yet.

State of Memory



After A is executed

Creating an Object Array - 2

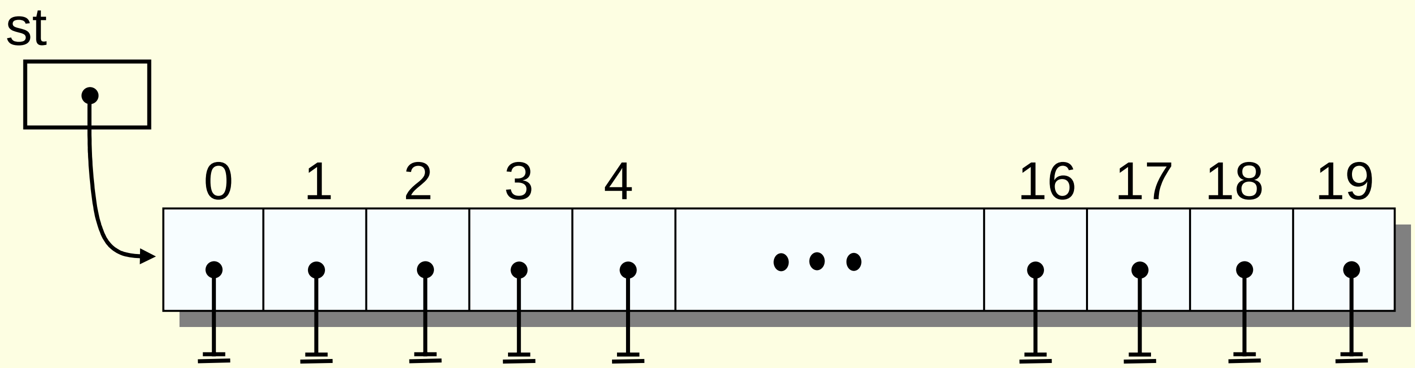
Code

B

```
Student[ ]  st;

st = new Student[20];

st[0] = new Student( );
```

Now the array for storing 20 Student objects is created, but the Student objects themselves are not yet created.

State of Memory



After B is executed

Creating an Object Array - 3

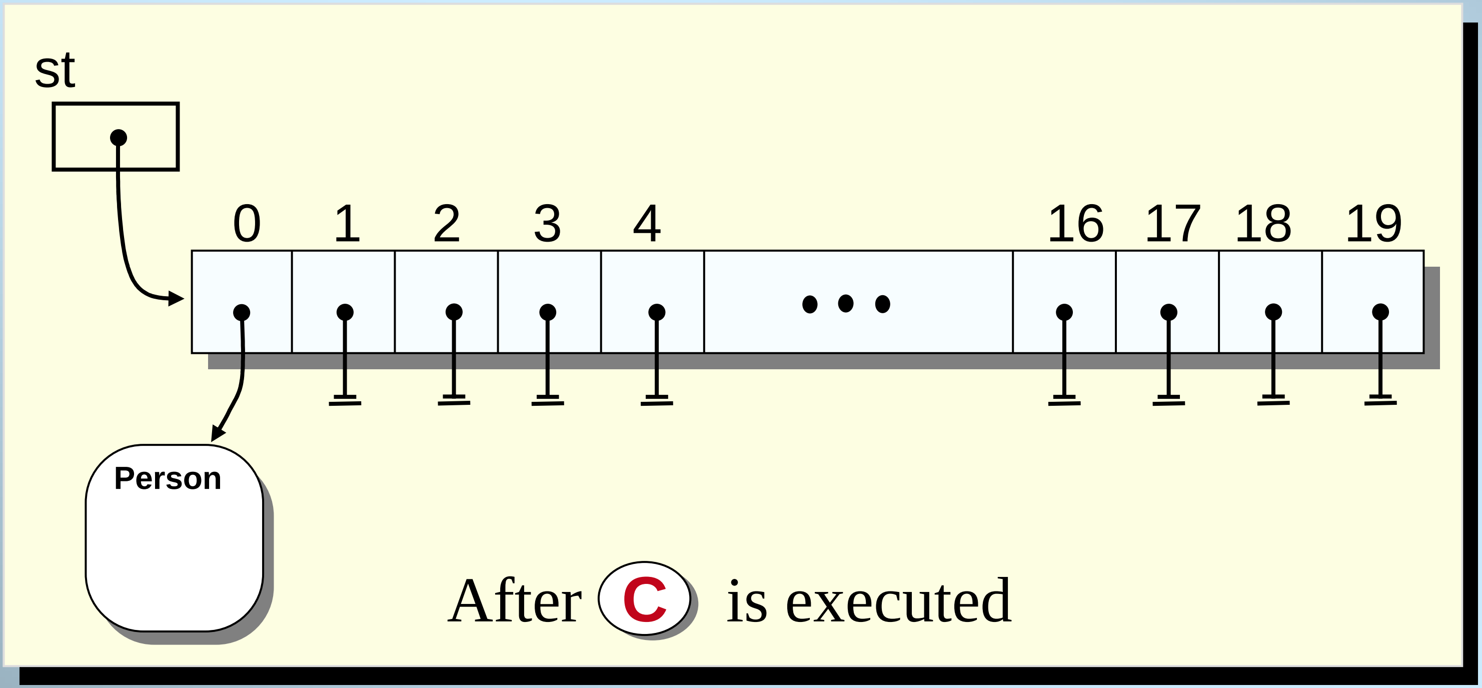
Code

C

```
Student[ ]  st;

st = new Student[20];

st[0] = new Student( );
```

One Student object is created and the reference to this object is placed in position 0.

State of Memory



After C is executed

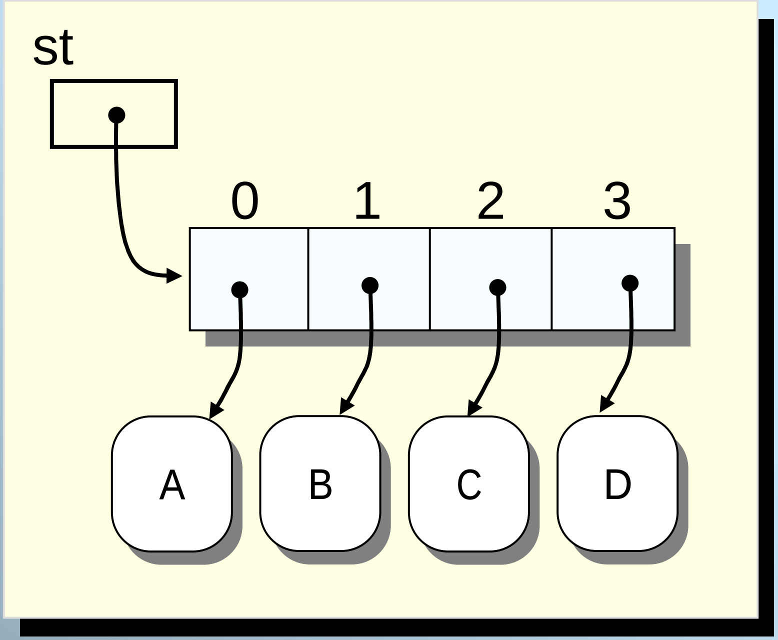
Object Deletion – Approach 1

```
int Idx = 1;
```

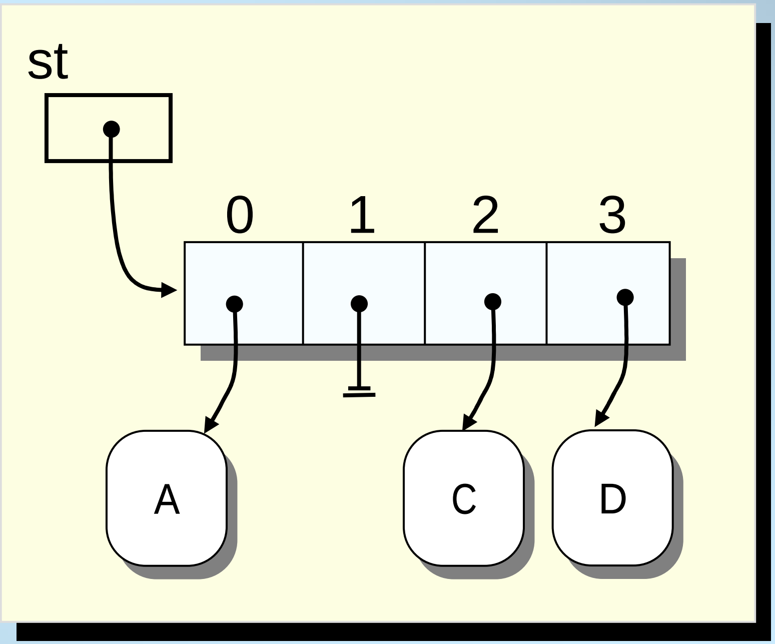
A

```
st[Idx] = null;
```

Delete Student B by setting the reference in position 1 to null.



Before A is executed



After A is executed

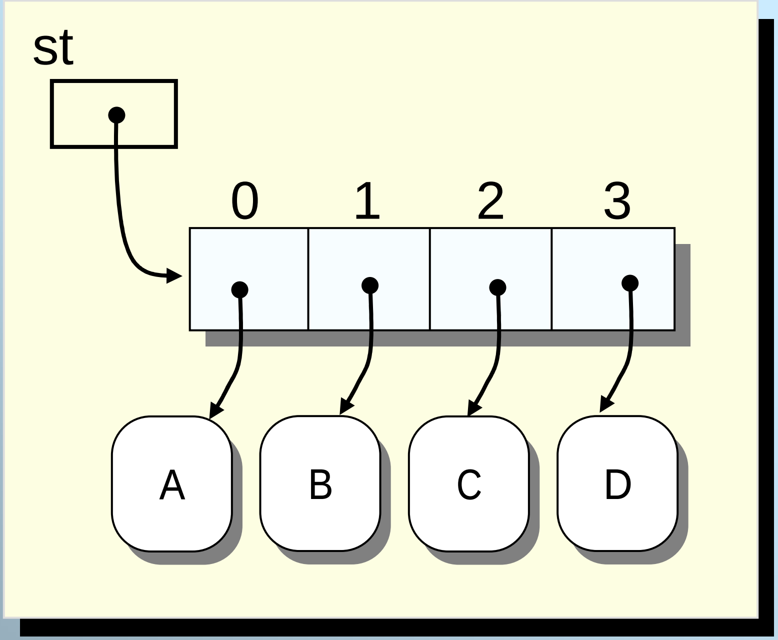
Object Deletion – Approach 2

```
int Idx = 1, last = 3;
```

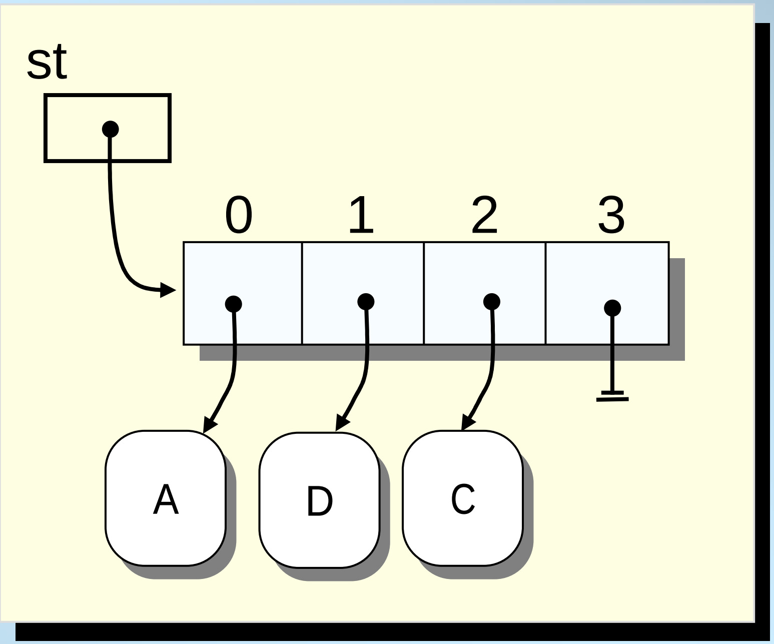
A

```
st[Idx] = st[last];

st[last]      = null;
```

Delete Student B by setting the reference in position 1 to the last person.



Before A is executed



After A is executed

Person Array Processing – Sample 1

Create Person objects and set up the p array.

```
import java.util.Scanner;

public class ArrayOfPersons {

private Person p[];

  private int nbp;
  Scanner input = new Scanner(System.in);

  public ArrayOfPersons(int size)

   {
      p = new Person[size];
      nbp=0;
    }

public ArrayOfPersons(Person pr[])
   {
      p = new Person[pr.length];
      for (int i =0; i< p.length; i++)
        p[i]= new Person(pr[i]); // p[i]=pr[i];
      nbp=p.length;
};

public void setArrayOfPersons(Person [] pr)
  {
    for (int i =0; (i< pr.length) && (i<p.length); i++)
      { p[i].setPerson(pr[i]); nbp++; }
  }
```

```
public void setArrayOfPersons()
   {  String s="";
       for (int i =0; i< p.length; i++)
     { p[i].setName(input.next()+input.nextLine());
        p[i].setAge(input.nextInt());
        s=input.next();
        p[i].setGender(s.charAt(0));
     }
     nbp=p.length;
   }
public boolean insertPerson(Person p1)
{ if (nbp = = p.length) return false;
   p[nbp++] = p1; // p[nbp]  = p1; nbp++;
   return true;
}
 //--- Average of all ages -----
   public double averageOfAge( )
   {   double s=0.0;
        for(int i =0; i<=nbp-1; i++)
         s+=p[i].getAge();
      return (s/nbp);
   }
```

Person Array Processing – Sample 1

Create Person objects and set up the person array.

```
//---- Find the oldest  persons
public Person OldestPerson()
{
    Person old = p[0];
    for (int i =1; i<=nbp-1; i++)
       if (old.getAge() < p[i].getAge())
           old =p[i];
    return (old);
}
  //--- search  for a particular person ----
public boolean findPersonByName(String na)
{
     for (int i=0; i<nbp; i++)  {
       if (p[i].getName().equals(na)== true)
             return (true);

     }
   return (false);
}
//---- return index of a person if exist and -1 if not
public int findPerson(Person pr)
{
      for (int i=0; i<nbp; i++)  {
        if (p[i].getName().equals(pr.getName())== true)
           if (p[i].getAge() == pr.getAge())
             if (p[i].getGender()== pr.getGender())
               return (i);

        }
      return (-1);
}
```

```
public  boolean delete1Person(Person pr)
{ int x = findPerson(pr)
   if (x != -1)
   {
      p[x] = p[nbp – 1];
     p[--nbp] = null;
      return true;
    }
    return false;
}


 public  boolean delete2Person(Person pr)
 { int x = findPerson(pr)
   if (x != -1)
   {for (int i = x; i<nbp - 1; i++)
     p[i] = p[i + 1];
     p[--nbp] = null;
     return true;
    }
    return false;
 }
}
```