CSE1030 – Introduction to Computer Science II

Lecture #15

Arrays

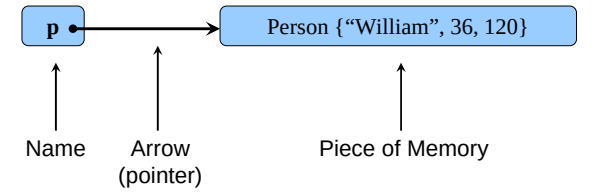
CSE1030 – Lecture #15

- Introduction to Arrays
- Constant Arrays Examples
- Dynamic Arrays
- We're Done!

Review: An Object is…

- An Object consists of three things:
  - A Name, an Arrow (Pointer), and Memory

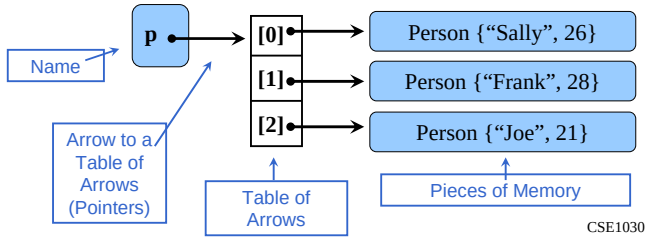
```
Person p = new Person("William", 36, 120);
```



New Idea: An Array is…

- A Name, and a **Table of Arrows (Pointers)**, to Blocks of Memory:

```
Person[] p = new Person[] {
   new Person("Sally", 26),
   new Person("Frank", 28),
   new Person("Joe", 21),
};
```



```
public class Person
{
   // attributes
   String Name;
   int    Age;

   // constructor
   Person(String name, int age)
      { Name = name; Age = age; }

   public String toString()
      { return Name + "(" + Age + ")"; }

   // methods
   String getName()       {return Name;}
   void setName(String n) { Name = n; }
   int getAge()           {return Age;}
   void setAge(int a)     { Age = a; }
}
```



```
class example1
{
   public static void main(String[] args)
   {
      Person[] p = {                          <-----
         new Person("Sally", 26),
         new Person("Frank", 28),
         new Person("Joe", 21),
      };

      System.out.println("Here's #2:");
      System.out.println( p[1] );             <-----

      System.out.println("Here's All of Them:");
      for(int i = 0; i < p.length; i++)       <-----
         System.out.println("  " + p[i]);     <-----

      System.out.println("Cause an Error! #4:");
      System.out.println( p[3] );
   }
}
```


Array Example Output

```
>java example1
Here's #2:
Frank(28)
Here's All of Them:
  Sally(26)
  Frank(28)
  Joe(21)
Cause an Error! #4:
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 3
        at example1.main(example1.java:20)
```


CSE1030 – Lecture #15

- Introduction to Arrays
- Constant Arrays Examples
- Dynamic Arrays
- We're Done!

Example: Days of the Week

- Write a program to convert:
  from the numeric "Day of the Week" (**`int`**)
  to the proper Day Names (**`String`**):

  1 → Monday
  2 → Tuesday
  3 → Wednesday
  4 → Thursday
  5 → Friday
  6 → Saturday
  7 → Sunday

Days of the Week – Poor Solution

```
import java.util.*;

class example2a
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      System.out.println("Enter a 'Day of the Week' "
                                 +"(1 <= integer <= 7):");

      int day_of_week = 0;
      try {
         day_of_week = in.nextInt();
      }
      catch(Exception e)
      {
         day_of_week = 0;
      }
```



```
      if(day_of_week < 1 || day_of_week > 7)
         System.out.println("Bad Input - try again");

      if(day_of_week == 1)
         System.out.println("Monday");
      else if(day_of_week == 2)
         System.out.println("Tuesday");
      else if(day_of_week == 3)
         System.out.println("Wednesday");
      else if(day_of_week == 4)
         System.out.println("Thursday");
      else if(day_of_week == 5)
         System.out.println("Friday");
      else if(day_of_week == 6)
         System.out.println("Saturday");
      else if(day_of_week == 7)
         System.out.println("Sunday");
   }
}
```


Days of the Week – Better Solution

```
import java.util.*;

class example2b
{
   public static void main(String[] args)
   {
      Scanner in = new Scanner(System.in);

      System.out.println("Enter a 'Day of the Week' "
                                 +"(1 <= integer <= 7):");

      int day_of_week = 0;
      try {
         day_of_week = in.nextInt();
      }
      catch(Exception e)
      {
         day_of_week = 0;
      }
```

```
      final String[] days = {
         "Monday", "Tuesday", "Wednesday", "Thursday",
         "Friday", "Saturday", "Sunday",
      };

      if(day_of_week < 1 || day_of_week > 7)
         System.out.println("Bad Input - try again");
      else
         System.out.println( days[day_of_week - 1] );
   }
}
```

Another Example – BaseballFielders 1

```
public class BaseballFielders
{
   private Person pitcher;
   private Person catcher;
   private Person firstBaseman;
   private Person secondBaseman;
   private Person thirdBaseman;
   private Person shortstop;
   private Person leftFielder;
   private Person centreFielder;
   private Person rightFielder;
```


One More Example – BaseballFielders 2

```
// constructor
public BaseballFielders(
   Person pitcher,
   Person catcher,
   Person firstBaseman,
   Person secondBaseman,
   Person thirdBaseman,
   Person shortstop,
   Person leftFielder,
   Person centreFielder,
   Person rightFielder
){
   this.pitcher       = pitcher;
   this.catcher       = catcher;
   this.firstBaseman  = firstBaseman;
   this.secondBaseman = secondBaseman;
   this.thirdBaseman  = thirdBaseman;
   this.shortstop     = shortstop;
   this.leftFielder   = leftFielder;
   this.centreFielder = centreFielder;
   this.rightFielder  = rightFielder;
}
```


One More Example – BaseballFielders 3

```
// accessors
public Person getPitcher()
   { return new Person(pitcher); }

public Person getCatcher()
   { return new Person(catcher); }

public Person getFirstBaseman()
   { return new Person(firstBaseman); }

public Person getSecondBaseman()
   { return new Person(secondBaseman); }

public Person getThirdBaseman()
   { return new Person(thirdBaseman); }

public Person getShortstop()
   { return new Person(shortstop); }

public Person getLeftFielder()
   { return new Person(leftFielder); }
```

One More Example – BaseballFielders 4

```
public Person getCentreFielder()
   { return new Person(centreFielder); }

public Person getRightFielder()
   { return new Person(rightFielder); }


// mutators
public void setPitcher(Person pitcher)
   { this.pitcher = new Person(pitcher); }

public void setCatcher(Person catcher)
   { this.catcher = new Person(catcher); }

public void setFirstBaseman(Person firstBaseman)
   { this.firstBaseman = new Person(firstBaseman); }

public void setSecondBaseman(Person secondBaseman)
   { this.secondBaseman = new Person(secondBaseman); }

public void setThirdBaseman(Person thirdBaseman)
   { this.thirdBaseman = new Person(thirdBaseman); }
```


One More Example – BaseballFielders 5

```
   public void setShortstop(Person shortstop)
      { this.shortstop = new Person(shortstop); }

   public void setLeftFielder(Person leftFielder)
      { this.leftFielder = new Person(leftFielder); }

   public void setCentreFielder(Person centreFielder)
      { this.centreFielder = new Person(centreFielder); }

   public void setRightFielder(Person rightFielder)
      { this.rightFielder = new Person(rightFielder); }

}
```


BaseballFieldersArray 1

```
public class BaseballFieldersArray
{
   private Person[] team;

   public static final int PITCHER       = 0;
   public static final int CATCHER       = 1;
   public static final int FIRSTBASEMAN  = 2;
   public static final int SECONDBASEMAN = 3;
   public static final int THIRDBASEMAN  = 4;
   public static final int SHORTSTOP     = 5;
   public static final int LEFTFIELDER   = 6;
   public static final int CENTREFIELDER = 7;
   public static final int RIGHTFIELDER  = 8;

   // constructors
   public BaseballFieldersArray()
      { team = new Person[9]; }
```


BaseballFieldersArray 2

```
public BaseballFieldersArray(
   Person pitcher,
   Person catcher,
   Person firstBaseman,
   Person secondBaseman,
   Person thirdBaseman,
   Person shortstop,
   Person leftFielder,
   Person centreFielder,
   Person rightFielder
){
   this();
   team[PITCHER]       = pitcher;
   team[CATCHER]       = catcher;
   team[FIRSTBASEMAN]  = firstBaseman;
   team[SECONDBASEMAN] = secondBaseman;
   team[THIRDBASEMAN]  = thirdBaseman;
   team[SHORTSTOP]     = shortstop;
   team[LEFTFIELDER]   = leftFielder;
   team[CENTREFIELDER] = centreFielder;
   team[RIGHTFIELDER]  = rightFielder;
}
```

BaseballFieldersArray 3

```
// copy constructor
public BaseballFieldersArray(BaseballFieldersArray other)
{
   team = new Person[9];
   for(int i = 0; i < 9; i++)
      team[i] = other.team[i];
}


// accessor
public Person getPlayer(int Player)
   { return team[Player]; }


// mutator
public void setPlayer(Person person, int Player)
   { team[Player] = person; }
```


BaseballFieldersArray - Testing

```
   // testing...
   public static void main(String[] args)
   {
      BaseballFieldersArray Players =
             new BaseballFieldersArray();

      Players.setPlayer(
             new Person("Henderson Alvarez", 22), PITCHER);
      Players.setPlayer(
             new Person("J.P. Arencibia", 26), CATCHER);

      Person theCatcher = Players.getPlayer(CATCHER);
      System.out.println("The catcher is: " + theCatcher);
   }
}
```


BaseballFielders Example Summary

- Array Solution is much shorter
  - 3 pages of code instead of 5
- and the Array Solution provides more functionality
  - Extra: no-parameter constructor
  - Extra: copy constructor
- Easier to Maintain the code
  - For example adding another field

Array Summary (1/2)

- Declare Arrays:

```
int[] someNumbers;
String[] words;
```

- Constructing Empty Arrays:

```
someNumbers = new int[10];
String[] words = new String[3];
```

- Initialising with Hardcoded Values:

```
int[] somenumbers = {
   2, 3, 5, 7, 11,
};
String[] words = {
   "Hello", "Good Bye"
};
```

Array Summary (2/2)

- Using Arrays:

```
String s = words[2];
int n = someNumbers[i];
somenumbers[3] = 17;
```

(Note: the Index must be an `int`)

- Array Size:

```
someNumbers.length
words.length
```


CSE1030 – Lecture #15

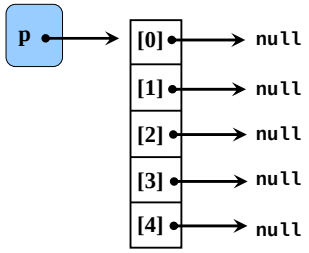
- Introduction to Arrays
- Constant Arrays Examples
- Dynamic Arrays
- We're Done!

What if we don't know how big an array needs to be, in advance?

- Start with a guess as to the correct size
- As objects are added or removed from the array, we need to **"Resize"** the array
- We can resize an array to make it larger, or smaller

Need to Store some "Person" Objects

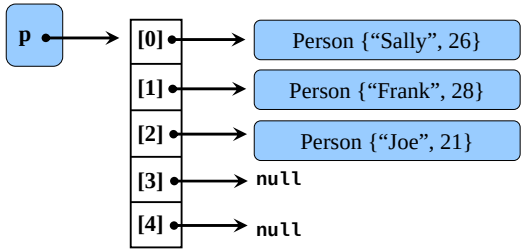
```
Person[] p = new Person[5];
int count = 0;
```



- "count" is:
  - the number of objects in the array
  - The index of the next available slot

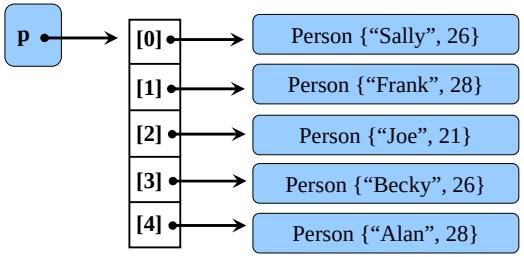
Put in some “Person” Objects

```
p[0] = new Person("Sally", 26);
p[1] = new Person("Frank", 28);
p[2] = new Person("Joe", 21);
count += 3;
```



What if we need to store 3 more?

```
p[3] = new Person("Becky", 26);
p[4] = new Person("Alan", 28);
p[?] = new Person("Jack", 21);
count += 2;
```

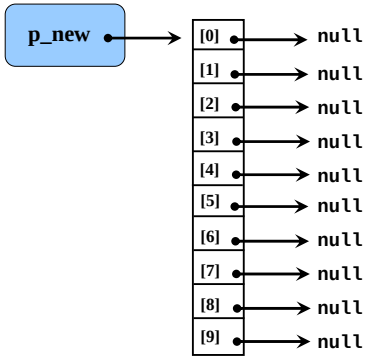


Resize – Larger

1. Need to create a new Larger array
2. Copy the objects over to the new array
3. Switch over to the new array

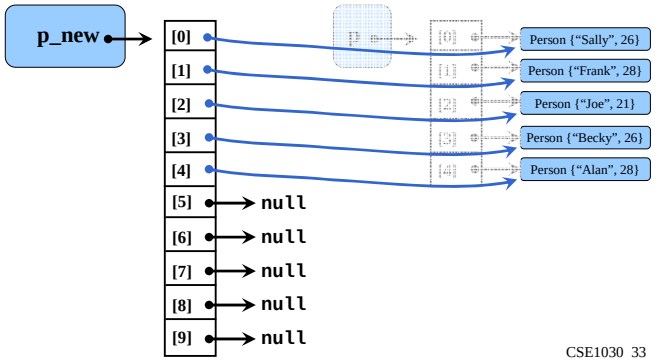
1 – Create a New Larger Array

```
Person[] p_new = new Person[p.length + 5];
```



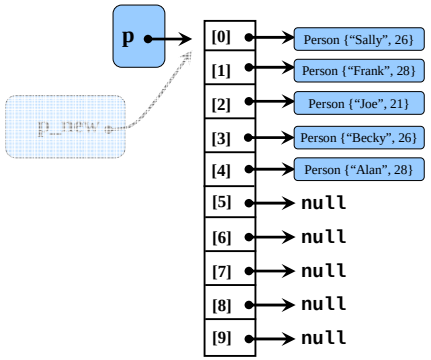
2 – Copy the objects over to the new array

```
for(int i = 0; i < p.length; i++)
   p_new[i] = p[i];
```



3 – Switch over to the new array

```
p = p_new;
```



Array Resize – Larger – Code Review

1. Need to create a new Larger array

```
Person[] p_new = new Person[p.length + 5];
```

(how much bigger?)

2. Copy the objects over to the new array

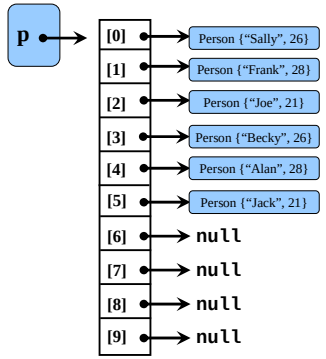
```
for(int i = 0; i < p.length; i++)
   p_new[i] = p[i];
```

3. Switch over to the new array

```
p = p_new;
```


Now we have room to add "Jack"

```
p[5] = new Person("Jack", 21);
count += 1;
```

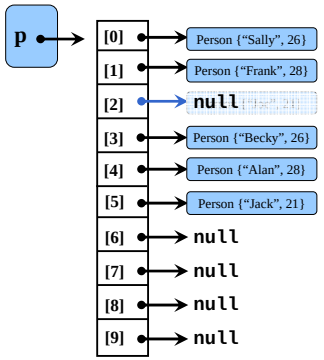


Next, Let's Delete

- Deletion is easy if it happens at the end of the list
- Deletion is less easy if it happens in the middle
- First, let's look at deleting from the middle of the table

Delete "Joe"

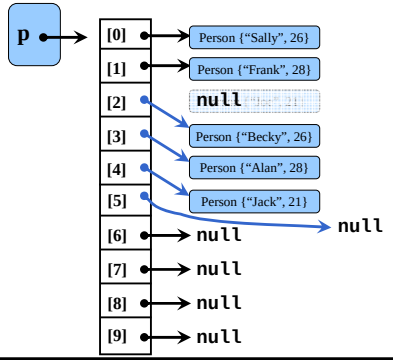
```
p[2] = null;
counter -= 1;
```



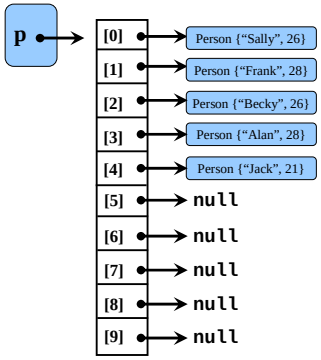
- Removing an object from the table is easy
- Are we storing "null" objects?
- Should we "shift" objects up to get rid of nulls in the table?

Shifting objects up to remove nulls

```
for(int i = 2; i < count; i++)
   p[i] = p[i+1];
p[count] = null;
```

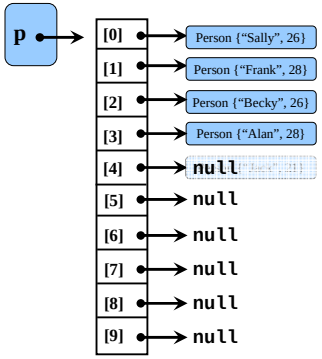


Final Result…



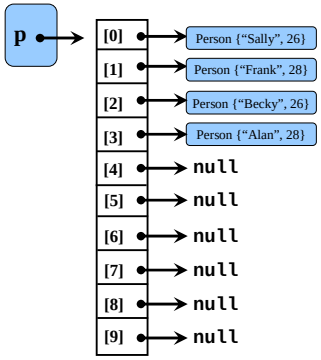
Next Delete "Jack" (from the End)

```
p[4] = null;
counter -= 1;
```



- Removing an object from the end of the table is easy
- Don't need to Shift objects up

Final Result…



Dynamic Arrays – Summary

- Three important Operations:
  - Adding
  - Deleting

  } These are tricky, because we may have to resize the array, or shift objects around – Inefficient!
  - Iterating

  } Fast, Fast, Fast! It's hard to beat `p[i]`
- Iterating is easy – just access what you want

```
for(int i = 0; i < count; i++)
   System.out.println(p[i]);
```


Arrays – The Big Questions

- Do we even allow the expensive operations (adding or removing to/from the middle of the list)
- Do we leave "null" values, or shift to remove them?
- Space / Time Trade-off:
  - How big an array do we start with?
  - By how many slots do we enlarge the array?
  - Do we ever shrink an array? By how much?

Efficient Operations

- Adding or Deleting to/from the End of the Array is fast (no Shifting), so those are safe operations to support, and to use
- Adding or Deleting to/from the Middle of the Array may require Shifting, which is inefficient
  - So maybe these operations should not be allowed?
  - Or only infrequently used?
  - Or they should be grouped together so all the shifting can be done at once?
  - Or we can leave "nulls", but that complicates things

Shifting Objects Up/Down

- There is a Trade-off:
  - Shifting to Remove null values:
    - Slower, but more memory efficient
    - Makes it easy to insert (**`count`** = available slot)
    - May not be so bad, if there are only a few deletes
  - Leaving nulls:
    - Means the user can't store nulls in the array
    - Faster Deletion
    - not necessarily faster Addition
      (searching for nulls is time consuming, changes order)
    - Can waste a Huge amount of memory
- In the end, it depends upon the properties of your data, and the requirements of your application. Experiment!!

CSE1030 – Lecture #15

- Introduction to Arrays
- Constant Arrays Examples
- Dynamic Arrays
- We're Done!

Next topic…

Arrays II