# CS 180 Problem Solving and Object Oriented Programming

Fall 2010

http://www.cs.purdue.edu/homes/apm/courses/CS180Fall2010/

Notes for Week 12:

Nov 8-12, 2010

Aditya Mathur
Department of Computer Science
Purdue University
West Lafayette, IN, USA

This Week:

11/8 Review

11/10

1. Concurrent programming
2. Concurrent linear search
3. CHALLENGE PROBLE
4. Concurrent bubble sort and merge

# Readings and Exercises for Week 12 [Preparation for Exam 2]

Readings:

Chapters:

6.3, 6.4, 6.5, 6.6, 6.7;

7.2, 7.3;

8.2, 8.3;

9.2, 9.3;

10.2, 10.3;

12.3.

Exercises:

Conceptual exercises at the end of chapters mentioned above.

# Special Sessions

Lab Help:

Mathur: Thursday November 11, 5:30-7:00pm
Alvin: Sunday Nov 14, 2-4pm

LWSN B158

Project 4 Due: Monday  November 15, 2010

# Lunch meeting

When: Based on appointments requested.

# Exam 2

BRING YOUR ID!

When: Tuesday November 9, 2010; 8-10pm.

Where: EE129

Format: Parts: A and B.

Part A: 30 points. Closed book/notes

15 multiple choice questions.

Part B: 70 points. Open book/notes

Two programming questions.

One question asks you to write methods.

Another question asks you to develop a simple GUI.

# Review: Arrays

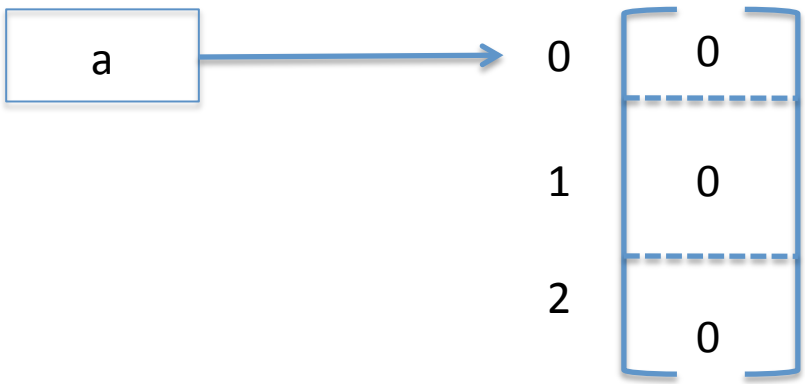
# Arrays of primitive types

- int [] a; // Declare an array
- int [] a=new int [n]; // Create an array

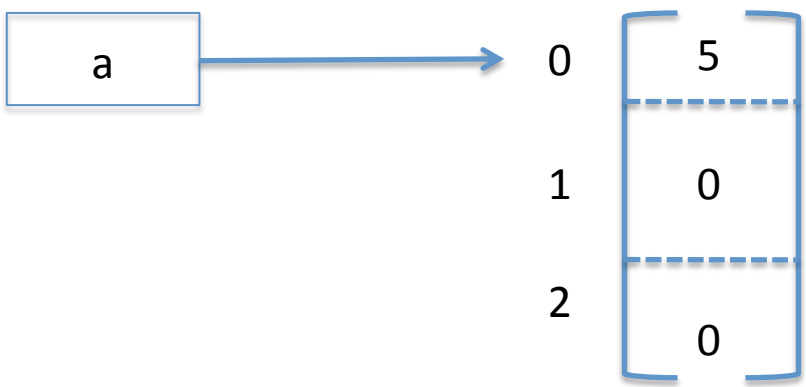
- long [][] x; // Declare an array
- long [][] x=new long [10][]; // Create an array
- long [][]x=new long [10][5]; // Create an array

# Array of primitive types: Visual

```
int [] a=new int [3]; // Create an array
```




```
a[0]=5;
```

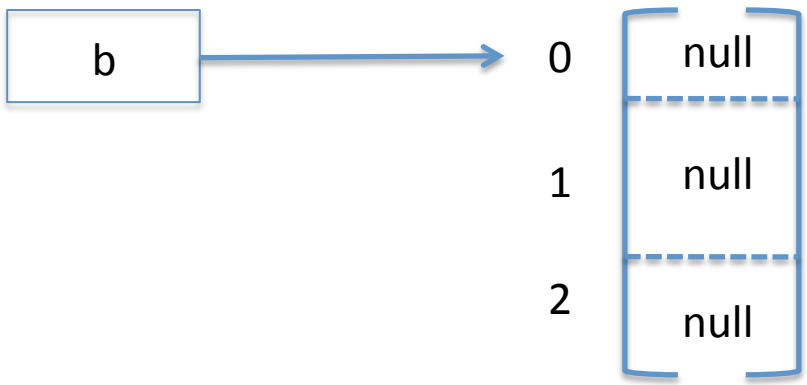


# Arrays of objects

- JButton [] b;
- JButton [] b=new JButton[n];

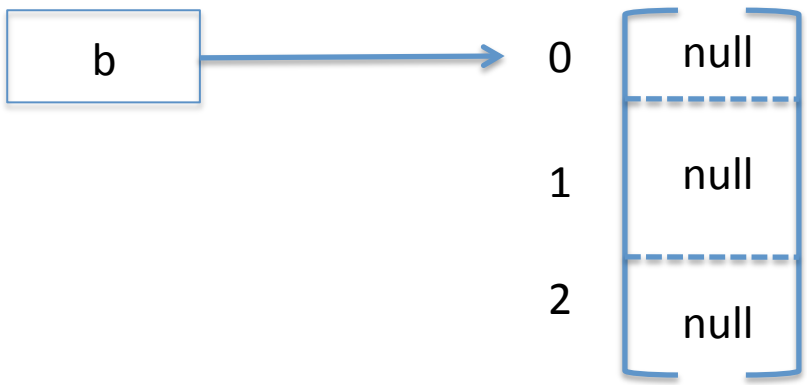
# Array of objects: Visual

```
JButton [] b=new JButton[3];
```



# Array of objects: Null pointer exception

```
JButton [] b=new JButton[3];
```

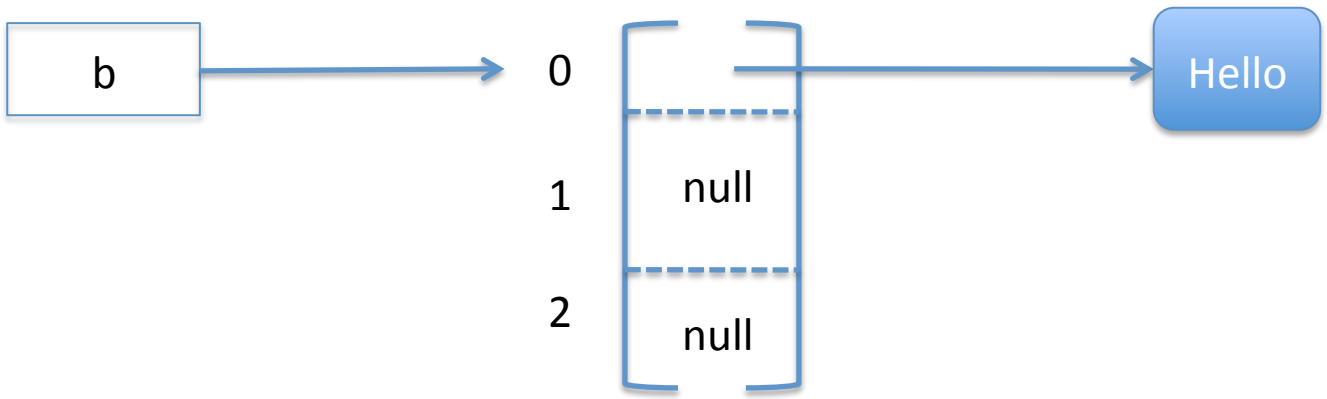



What happens if we do the following?

```
button[0].setText("Hello");
```

# Array of objects: Creating an object

```
JButton [] b=new JButton[3]
b[0]=new JButton("Hello");
```

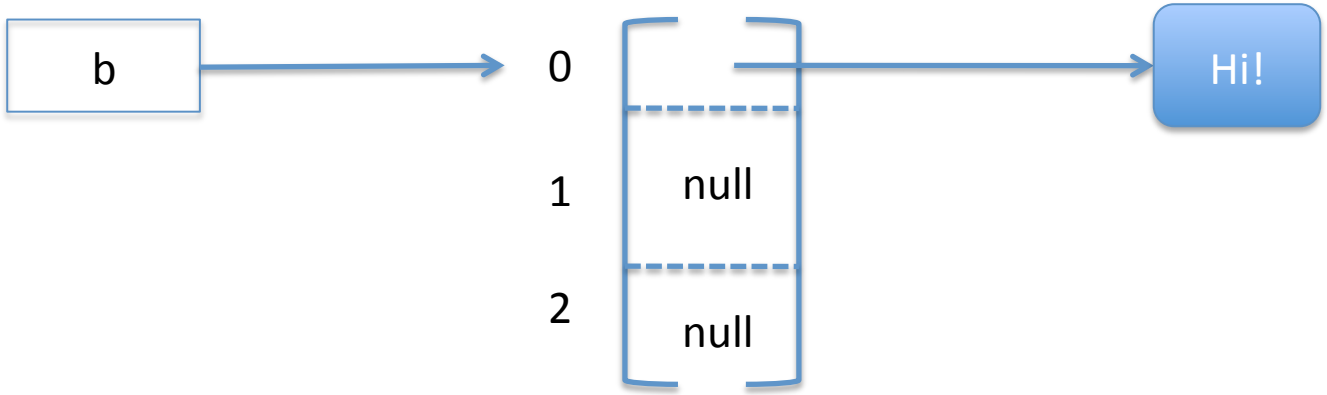


# Array of objects: Modifying an object

```
JButton [] b=new JButton[3]
b[0]=new JButton("Hello");
```

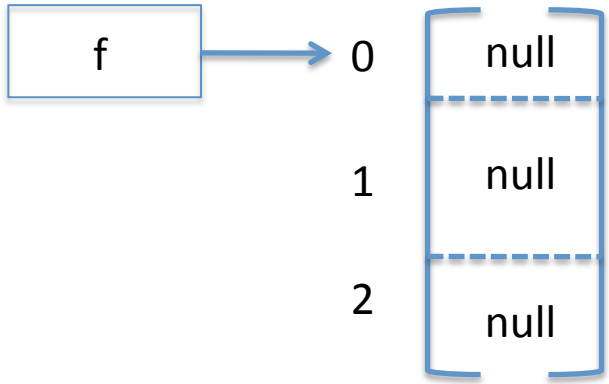
What happens if we do the following?

```
b[0].setText("Hi!");
```

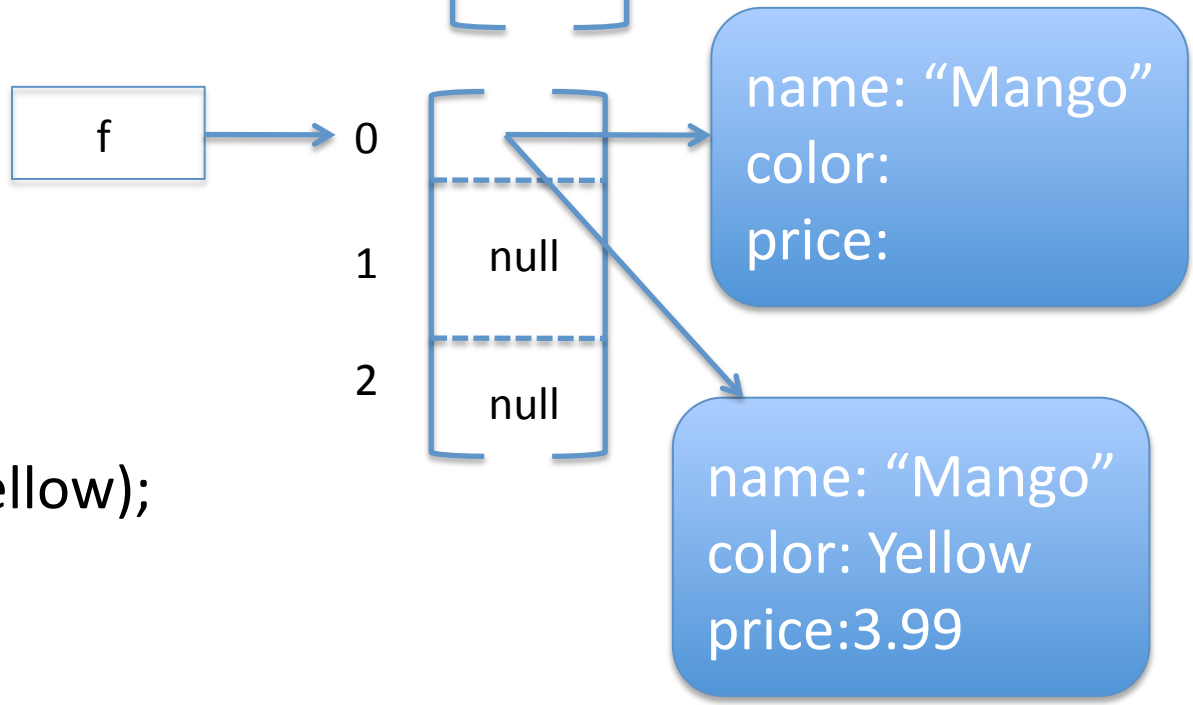


# Array of objects: Another example

Fruit [] f=new Fruit[3]




f[0]=new Fruit("Mango");




f[0].setFruitColor(Color.Yellow);

f[0].setPrice(3.99);

# Review: Methods

# Method header

Modifier public/ private/ none

Modifier static/ none

Return type

Name

Parameters 0 or more

public boolean find(int [] a, int x)

# Method parameters

Reference parameter

Value parameter

```
public   boolean   find(int [] a, int x)
```

Value parameter: primitive types

Reference parameter: objects

# Value parameters

Value of actual parameter is passed

```
public   boolean   find(int [] a, int x){

     System.out.println(x);
     x++;
}
```

What is displayed?

```
int y=15;
int[] num={2, 3, 15, -29};
boolean found=find(num, y)
System.out.println(found + ” “+ y);
```

# Reference parameters

Reference to actual parameter is passed

```
public   boolean   find(int [] a, int x){
     System.out.println(a[1]);
     x++;
     a[1]++;
}
```

What is displayed?

```
int y=15;
int [] num={2, 3, 15, -29};
boolean found=find(num, y)
System.out.println(found + ” “+ num[1]);
```

# The main() method

Is this value or a reference parameter?

```
public static void main(String [] args){

}
```

How can we pass parameters to the main() method?

# Method parameter: summary

- A method may have zero or more parameters.
- Each parameter has a name.
- Each parameter must have  a type
- All parameters are passed by value. For primitive types the value of the actual parameter is passed. For reference types a reference to the object is passed.
- Each method must have a return type.
- Each parameter becomes a local variable for the method.
- A constructor is a special method that has no return type.

# Review: Class and instance variables

# Class and instance variables

```
public class Home{

public static String measureUnit="Square Feet";
public static String code="Pub. L. 110-140";
private  String address="";
private int bedRooms=4;

public Home(String a, int r){
     address=a;
     bedRooms=r;
     }
public int getBedrooms(){
     return(bedRooms);
}
```

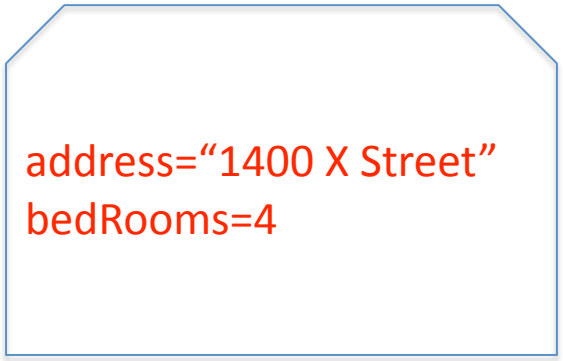
Class variables

Instance variables

# Class and instance variables

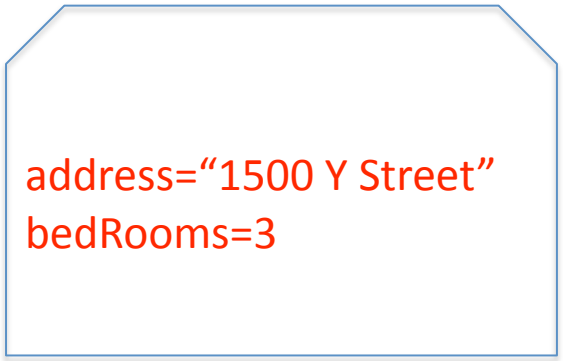
```
Home h1=new Home("1400 X Street", 4);

System.out.println(h1.getBedRooms());
```

h1




```
Home h2=new Home("1500 Y Street", 3);
System.out.println(h2.getBedRooms());
```

h2




```
System.out.println(h1.code);
System.out.println(h2.code);
System.out.println(Home.getBedRooms());
System.out.println(Home.code);
```

# Class variables

Declared using the static keyword.

Not a part of the object, but part of the class in which they are declared.

Accessible via the objects created from that class

Declare a variable or a method as static only if does not depend on the object.

# Instance variables

Declared without using the static keyword

Part of an object

Accessible via the objects created from the parent class

Declare a variable as an instance variable if its value depends on the object.

# The main() method

Is declared to be static

Can it access instance variables in its parent class?

What variables and methods in the parent class can the main method access?

# Accessibility rules

If a variable or an object declaration uses this modifier

then can this variable or object be used inside ?

Y: Yes.
N: No.

| Modifier | Class | Package | Subclass | World |
|---|---|---|---|---|
| public | Y | Y | Y | Y |
| private | Y | N | N | N |
| none | Y | Y | N | N |

Package: A collection of classes identified as a Java package.
World: Collection of packages

# Review: Local and global variables

# Local variables

```
public void test(int x){

int p, q;

}
```

x, p, q are local to method test.

test is public and hence global and can be used in all classes.

Variables and objects declared inside a method are local to that method.

# Local variables: Use of public and private modifiers

```
public void test(int x){

int p, q;
public int z;

}
```

x, p, q are local to method test.

test is public and hence global and can be used in all classes.

Declarations inside a method cannot be preceded by public or private.

# Global variables

```
public class  Toy{

public String name;
private String owner;

}
```

name is global as it is declared to be public.

Owner is local to this class

Toy is global and can be used anywhere.

# Local and global variables: Example

plus can
be used here

```
public class X{

public JButton plus;

public void doSomething(String c){
    int z;
    for (int i=0; i<10; i++){
        int p;
        }
    System.out.println(i, p);
    } // end of method
}// end of class
```

i can be
Used here

z, c can
be use here

i, p out of scope

# Review: Inheritance

# Inheritance

```
public class Fruit{

public String name;
private String color="Red";
int type;

public void
    changeColor(String c){
        Color=c;
    }
public String getColor(){
        return color;
    }
}
```

```
public class Mango extends Fruit{

public String origin;

public void harvest(){

    }
public String getOrigin(){
        return color;
    }
}
```

All methods and local variables/ objects are available to Mango.

# Inheritance: Another example

```
public class Gui extends JFrame{

}
```

All methods and local variables/ objects of JFrame are available to Gui.

# Review: GUI

# GUI: Widget/Methods

| | |
|---|---|
| JFrame | setSize() |
| JPanel | |
| JButton | setVisible( ) |
| JTextField | |
| JMenuBar | setText() |
| JMenu | |
| JMenuItem | getText |
| | add() |

# GUI: Listeners/Methods

| | |
|---|---|
| ActionListener | getSource() |
| MenuListener | |
| KeyListener | addActionListener() |
| MouseListener | |

# GUI: Interface

A class implements an interface

All methods in the interface must be implemented.

ActionListener is an interface.

# GUI: Abstract Class

Similar to interface but may implement some methods thus avoiding the need to implement all methods.

MouseAdapter is an abstract class.

# Back to Concurrency

# Thread: What is it?

- A thread is a sequence of computation that can run in parallel with other threads.
- Every program has at least one thread of computation.
- Each thread is associated with an instance of the class Thread.
- A program with two or more threads is generally referred to as a multithreaded program.
- A multithreaded program is generally used to speed up the solution to a problem.

# Thread: Typical lifecycle

Define class that extends Thread.

↓

Create thread.

↓

Start thread.

↓

Wait for thread to terminate.

↓

Get results from thread.

↓

Use results from thread.

# Thread: Defining a class

Thread is a class. One way to create a new thread is to first define a class that extends the Thread class.

```
public class Search extends Thread{
    String [] x; // For use by a thread
    String s; // Another object for use by the thread
    int tID; //  Thread ID if needed
    int start, end; // Start and end indices for search
    boolean found; // Computed by the thread
}
```

# Thread: The constructor

A class that extends Thread generally has a constructor that is used to pass parameters to a thread as follows.

```
public Search (String  [] a, String s, int start, int end, int tID, ){
// Save parameters for use when the thread executes
    x=a;
    this.s=s; this.start=start; this.end=end;
    this.tID=tID
}
```

# Thread: The run() method

Unlike the main() method, every thread object must have a run() method.

When a thread is started, its execution begins in the run() method and terminates when the run() method terminates

# Thread: The run() method: Example

```
public void run (){  // Must not have any parameters
// Thread execution begins here.
    System.out.println("Hi! I am thread "+tID);
    System.out.println("Bye bye! See you!");
}
```

The run() method does not take any parameters.

The run() methods does not return any value.

# Thread: Other methods

A thread may have methods other than the run() method. Here is an example.

```
public boolean getOutcome(){

    return (found); // Return the outcome of search
}
```

Methods in a thread may be called during the execution of the run() method and even after the completion of the run() method.

# Thread: run() method

A run() method is like any other method except that it does not have any parameters or return type.

A run() method may call any other method to complete its task.

# Thread: Creation

Create a thread object just as you would for any other object. The following examples create two thread objects named one and two.

```
Search one=new Search (a, s, start, end, 1);
Search two=new Search (a, s, start, end, 2);
```

# Thread: Start of execution

Execution of a thread must be started using the start() method for that thread. Here are two examples for starting threads one and two.

```
one.start(); // Starts the execution of thread object one
two.start(); // Starts the execution of thread object two
```

# Thread: Waiting for completion

You may wait for a thread to complete execution using the join() method as follows.

```
try{
    one.join(); // Wait for thread one to complete
    two.join(); // Wait for thread one to complete
catch(Exception e){}
```

The try-catch block is needed. More on this later!

# Thread: Extracting results

You may extract the outcome of a thread's execution in a variety of way. One way is to use an accessor method to do so. Here are examples.

```
boolean f1=one.getOutcome(); // Get search outcome of thread one
boolean f2=two.getOutcome(); // Get search outcome of thread two
```

# Thread: Typical lifecycle [Review]

Define class that extends Thread.

↓

Create thread.

↓

Start thread.

↓

Wait for thread to terminate.

↓

Get results from thread.

↓

Use results from thread.

# Example: Concurrent search

# Problem 1

We are given a rather large  array of strings. We wish to write a program that will determine, and display, whether or not a given string appears in the array.

Given that the array is large, we are required to write a concurrent program that uses threads to solve the problem faster than it would if done sequentially.

# Understanding the problem

The problem is rather straightforward!

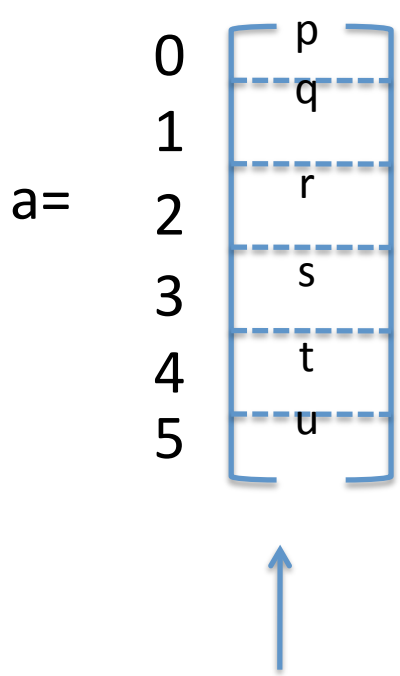
# Search algorithm

There are many search algorithms available. Given no information about how the array of strings is ordered, we will use a simple linear search algorithm.

The given string will be compared with the first element of the array, then the next and so on until a match is found or all elements have been compared.

If a match is found, the algorithm returns true, otherwise it returns false.

# Concurrent search: Create threads




Given array

Create thread one

Pass
a, s and start and end indices to the constructor

[start=0, end=2]

Create thread two

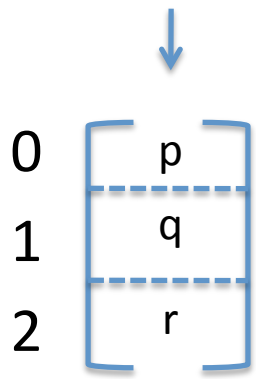
Pass
a, s and start and end indices to the constructor

[start=3, end=5]

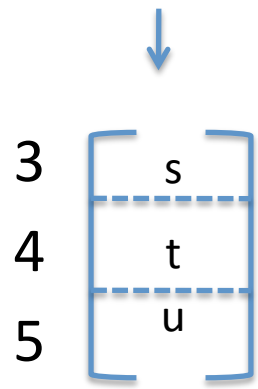
Given string to be searched: s

# Concurrent search: Array partitioned

Thread one searches in this part of the array




Thread two searches in this part of the array



# Concurrent search: Next steps

- Start the threads;
- Wait for the threads to complete;
- Extract the search outcome of each thread;
- If any one of the threads found the string then the search was successful otherwise the search failed.

Live demo

# Challenge Problem

You are given an array of integers. The integers in the array vary from 0 (inclusive) to 500 (inclusive).

You are also given 3 bins.

Write a concurrent program that distributes the integers in the array into the 3 bins such that bin 1 gets all integers less than 170, bin 2 gets all integers greater than 170 but less than 240, and bin 3 gets all integers greater than 240.

Sort the bins in ascending order and display.

# Challenge Problem: Example

Given:

array={3, 43, 129, 32, 400, 452, 5}

Final contents of the three bins as displayed:

bin 1={3, 5, 32, 43}
bin 2={129}
bin 3={400, 452}

Week 12: November 8-13, 2010
Hope you enjoyed this week!

Questions?

Contact your recitation instructor. Make full use of our office hours.