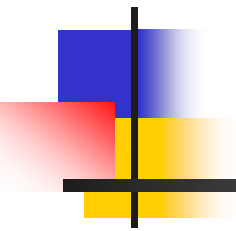
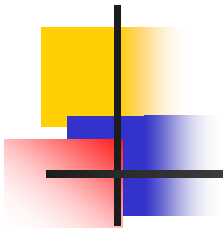


Arrays 3/4



Outline

- Arguments for the Method **`main`**
- Methods That Return an Array
- Partially Filled Arrays
- Accessor Methods Need Not Simply Return Instance Variables
- Privacy Leaks with Array Instance Variables
- Example

- Arguments for the Method **`main`** …

- The heading for the **`main`** method of a program has a parameter for an array of **`String`**
  - It is usually called ***`args`*** by convention

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

  - Note that since ***`args`*** is a parameter, it could be replaced by any other non-keyword identifier
- If a Java program is run without giving an argument to **`main`**, then a default empty array of strings is automatically provided

... - Arguments for the Method **`main`** ...

- Here is a program that expects three string arguments:

```
public class SomeProgram
{
  public static void main(String[] args)
  {
    System.out.println(args[0] + " " + args[2] + args[1]);
  }
}
```

- Note that if it needed numbers, it would have to convert them from strings first

... - Arguments for the Method `main`

- If a program requires that the **`main`** method be provided an array of strings argument, each element must be provided from the command line when the program is run

  ```
  java SomeProgram Hi ! there
  ```

  - This will set **`args[0]`** to "Hi", **`args[1]`** to "!", and **`args[2]`** to "there“
  - It will also set **`args.length`** to 3

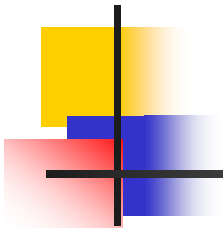
- When **`SomeProgram`** is run as shown, its output will be:

  ```
  Hi there!
  ```

- Methods That Return an Array

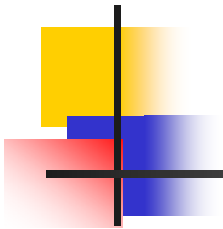
- In Java, a method may also return an array
  - The return type is specified in the same way that an array parameter is specified

```
public static int[]
    incrementArray(int[] a, int increment)
{
  int[] temp = new int[a.length];
  int i;
  for (i = 0; i < a.length; i++)
    temp[i] = a[i] + increment;
  return temp;
}
```



- Partially Filled Arrays ...

- The exact size needed for an array is not always known when a program is written, or it may vary from one run of the program to another
- A common way to handle this is to declare the array to be of the largest size that the program could possibly need
- Care must then be taken to keep track of how much of the array is actually used
  - An indexed variable that has not been given a meaningful value must never be referenced

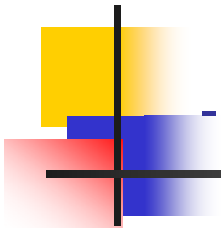


... - Partially Filled Arrays

- A variable can be used to keep track of how many elements are currently stored in an array
  - For example, given the variable ***count***, the elements of the array **someArray** will range from positions

    someArray[0] through **someArray[*count* - 1]**

  - Note that the variable **count** will be used to process the partially filled array instead of **someArray.length**
  - Note also that this variable (**count**) must be an argument to any method that manipulates the partially filled array



- Accessor Methods Need Not Simply Return Instance Variables

- When an instance variable names an array, it is not always necessary to provide an accessor method that returns the contents of the entire array
- Instead, other accessor methods that return a variety of information about the array and its elements may be sufficient

- Privacy Leaks with Array Instance Variables ...

- If an accessor method does return the contents of an array, special care must be taken
  - Just as when an accessor returns a reference to any private object

```
public double[] getArray()
{
  return anArray;//BAD!
}
```

  - The example above will result in a *privacy leak*

... - Privacy Leaks with Array Instance Variables...

- The previous accessor method would simply return a reference to the array **anArray** itself
- Instead, an accessor method should return a reference to a *deep copy* of the private array object
  - Below, both **a** and **count** are instance variables of the class containing the **getArray** method

```
public double[] getArray()
{
  double[] temp = new double[count];
  for (int i = 0; i < count; i++)
    temp[i] = a[i];
  return temp
}
```

... - Privacy Leaks with Array Instance Variables

- If a private instance variable is an array that has a class as its base type, then copies must be made of each class object in the array when the array is copied:

```
public ClassType[] getArray()
{
  ClassType[] temp = new ClassType[count];
  for (int i = 0; i < count; i++)
    temp[i] = new ClassType(someArray[i]);
  return temp;
}
```

- Example ...

```
/**
 Class for a partially filled array of doubles. The class enforces the
 following invariant: All elements are at the beginning of the array in
 locations 0, 1, 2, and so forth up to a highest index with no gaps.
*/
public class PartiallyFilledArray
{
    private int maxNumberElements; //Same as a.length
    private double[] a;
    private int numberUsed; //Number of indices currently in use

    /**
     Sets the maximum number of allowable elements to 10.
    */
    PartiallyFilledArray()
    {
        maxNumberElements = 10;
        a = new double[maxNumberElements];
        numberUsed = 0;
    }

    /**
     Precondition arraySize > 0.
    */
    PartiallyFilledArray(int arraySize)
    {
        if (arraySize <= 0)
        {
            System.out.println("Error Array size zero or negative.");
            System.exit(0);
        }
        maxNumberElements = arraySize;
        a = new double[maxNumberElements];
        numberUsed = 0;
    }
    PartiallyFilledArray(PartiallyFilledArray original)
    {
        if (original == null)
        {
            System.out.println("Fatal Error: aborting program.");
            System.exit(0);
        }                                                    (continued)
```

... - Example ...

```
        maxNumberElements =
                  original.maxNumberElements;
        numberUsed = original.numberUsed;
        a = new double[maxNumberElements];
        for (int i = 0; i < numberUsed; i++)
            a[i] = original.a[i];
    }

    /**
     Adds newElement to the first unused array position.
    */
    public void add(double newElement)
    {
        if (numberUsed >= a.length)
        {
            System.out.println("Error: Adding to a full array.");
            System.exit(0);
        }
        else
        {
            a[numberUsed] = newElement;
            numberUsed++;
        }
    }

    public double getElement(int index)
    {
        if (index < 0 || index >= numberUsed)
        {
            System.out.println("Error:Illegal or unused index.");
            System.exit(0);
        }

        return a[index];
    }
    /**
     index must be an index in use or the first unused index.
    */
    public void resetElement(int index, double newValue)
```

Note that the instance variable a is a copy of `orignal.a`. *The following would not be correct:*
`a = original.a;`
This point is discussed in the subsection entitled "Privacy Leaks with Array Instance Variables."

... - Example ...

```
{
     if (index < 0 || index >= maxNumberElements)
     {
          System.out.println("Error:Illegal index.");
          System.exit(0);
     }
     else if (index > numberUsed)
     {
          System.out.println(
                    "Error: Changing an index that is too large.");
          System.exit(0);
     }
     else
          a[index] = newValue;
}

public void deleteLast()
{
     if (empty())
     {
          System.out.println("Error:Deleting from an empty array.");
          System.exit(0);
     }
     else
          numberUsed--;
}

/**
 Deletes the element in position index. Moves down all elements with
 indices higher than the deleted element.
*/
public void delete(int index)
{
     if (index < 0 || index >= numberUsed)
     {
          System.out.println("Error:Illegal or unused index.");
          System.exit(0);
     }

     for (int i = index; i < numberUsed; i++)
          a[i] = a[i + 1];
     numberUsed--;
}
```

... - Example

```
    public boolean empty()
    {
        return (numberUsed == 0);
    }

    public boolean full()
    {
        return (numberUsed == maxNumberElements);
    }

    public int getMaxCapacity()
    {
        return maxNumberElements;
    }

    public int getNumberOfElements()
    {
        return numberUsed;
    }
}
```

THE END