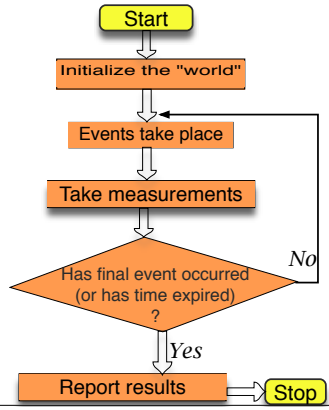
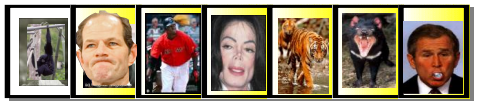
Simulation

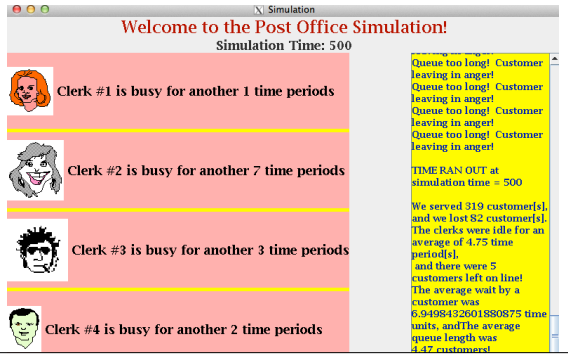


181

The Queue Class



PostOffice Simulation



183

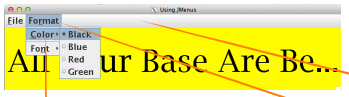
Animation (see Animate.java)

- The abstract class Image is the superclass of all classes that represent graphical images. You cannot create an object of class Image; instead you request that an Image be loaded (from a local file or a URL) via the ImageIcon class:

```
static ImageIcon [] mouse;
mouse = new ImageIcon [15];
for (int i=0; i < 15; i++)
  mouse[i] = new ImageIcon ( "T" + (i+1)  + ".gif");
```

- You can also use method `paintIcon(Component c, Graphics g, int x, int y)` to draw the individual images on screen. See ScratchAnimation.java

Menus (see MenuTest.java)



- Menus allow a program to grow without cluttering the interface.
- A **JMenuBar** is a horizontal list of menus.
- A **JMenu** is a clickable area on the menu bar that is associated with a **JPopupMenu**, a small window that displays **JMenuItem**s.
- **JSeparator**s are used to group menu items.

Menu Example (MenuTest.java)

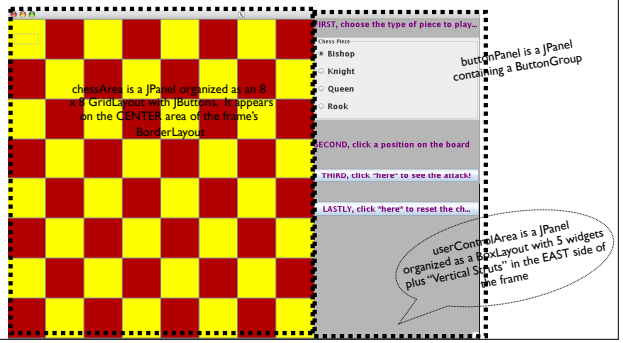
```
JMenuBar bar = new JMenuBar();                    // create menubar
setJMenuBar( bar );                               // set the menubar for the JFrame
                                                  // create File menu and Exit menu item
JMenu fileMenu = new JMenu ( "File" );
fileMenu.setMnemonic ( 'F' );
JMenuItem aboutItem = new JMenuItem( "About..." );
aboutItem.setMnemonic ( 'A' );
aboutItem.addActionListener (
  new ActionListener() {
    public void actionPerformed( ActionEvent e )
     {  JOptionPane.showMessageDialog( MenuTest.this,
          "This is an example\nof using menus",
          "About", JOptionPane.PLAIN_MESSAGE );
     }
                        }     );
fileMenu.add (aboutItem );
JMenuItem exitItem = new JMenuItem( "Exit" );
```

Keyboard equivalent for specific menus and menu items

Action listeners are associated with menu items.

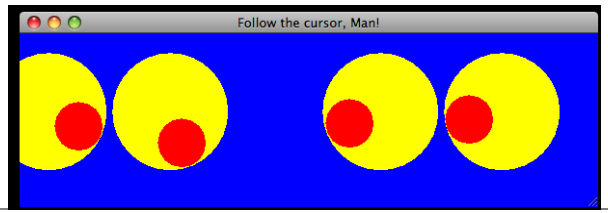
JMenuItem is added to a JMenu

Chessboard GUI



Eyes that Follow the Mouse

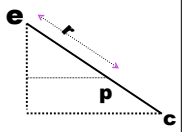
- EyeDemo.java creates a JPanel that contains an instance variable (cursor) that contains the location of the mouse, and two instance variables of type PairOfEyes (e1 and e2).



Eyes that Follow the Mouse

- Consider the distance **d** from point e to point c is $\sqrt{(c.x - e.x)^2 + (c.y - e.y)^2}$

- We also know r is the radius of the eye. Using similar triangles, we see the center of the pupil is at (p.x, p.y), where
  p.x= e.x+ (c.x-e.x) * r/d and
  p.y = e.y+ (c.y-e.y) * r/d
  (Actually, r is EYE_RADIUS-PUPIL_RADIUS .)



See PairOfEyes.java and EyeDemo.java

Sets

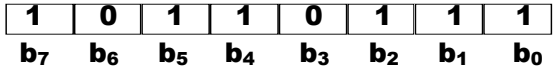
- In mathematics, an unordered collection of distinct elements is called a set.
  - Elements can be added, located and removed.
  - Sets don't have duplicates.
  - The order of the objects doesn't really matter (unlike an array or an ArrayList)
- In Java, sets are implemented both as "hash tables" and as "trees." Both HashSet and TreeSet implement the Set interface.

Defining a Set Class in Java

- By defining set objects as a class we can use sets as an *abstract data type* in our Java programs, without having to worry about implementation details! See file TestSets.java The main program loop looks roughly like this:
  - case 1: … setA.readSet(); … break;
  - case 2: … setB.readSet(); … break;
  - case 3: … System.out.print (setA.intersect (setB)); …
  - case 4: … System.out.print (setA.union (setB)); …
  - case 5: … System.out.print (setA.difference (setB)); …
  - default: …
- Additional methods for adding and removing a single element from a set (among others) are also defined (in addition to *union*, *difference*, and *intersect*).

Representing a Set Using a Single Byte

- A byte (8 bits) is the smallest addressable unit of main memory. The same byte (bit pattern) can represent various things:



  - an unsigned integer
  - a signed integer
  - an ASCII character
  - the SET whose elements are …

Basic Set Operations

- Set A = {2, 3, 4, 7} A = 1 0 0 1 1 1 0 0

  Set B = {2, 4, 5} B = 0 0 1 1 0 1 0 0

  0 0 0 1 0 1 0 0

- The UNION of the two sets **(A + B)** consists of all elements that are in either A or B. In Java, the operation on bytes can be expressed as …
- The INTERSECTION of the two sets **(A * B)** consists of all elements that are in both A and B. In Java, this operation can be expressed as …

Set Operations, *continued*

- The SYMMETRIC DIFFERENCE of two sets consists of elements that are in one set but not the other. In our example, this is {3, 5, 7}. This is expressed as …
- The COMPLEMENT of a set consists of all elements not in the set. The complement of b can be expressed in Java as …
- The DIFFERENCE of two sets (A - B) consists of elements that are in A but not in B. In Java this operation is …
- In Java there are bitwise operations that incorporate assignment: **|=**, **&=**, **^=**
- These are useful for setting, clearing, or complementing a single bit within a byte. For example,
  - `a |= (1 << n)` //sets what???

Instance Data Members

- To have sets of a fixed size (say, those whose elements range from 0 to 255, requiring 256/8 = 32 bytes):

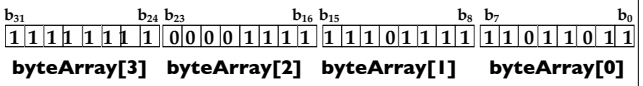
```
class Bitset
{  private byte byteArray[]= new byte[32];
   …
}
```

- But to have sets of different sizes, use

```
class Bitset
{   private int maxSize;
    private byte [] byteArray;
    …
```

Representing Larger SETs

- Use nbyte consecutive bytes of memory to represent a set with 8*nbyte possible members.
  - For example, with 4 bytes we can have members from 0 to 31:



  - To find the byte in which element n is represented, the address we need is **byteArray[n/8]** and the bit number at that address is …

- Now we can implement primitives like

```
void setBit (int n)
{  int whichByte = n / 8;
    int whichBit = n % 8;
    byteArray[whichByte]  |=  (1<< whichBit);
}
```


More Primitive Operations

- See file Bitset.java for the details:

```
boolean getBit ( int n )          // Returns the n'th bit value
{    int whichByte = n / 8;
     int whichBit = n % 8;
     return ( (byteArray [whichByte] &  (1 << whichBit) )  != 0 );
}

void clearBit ( int n )                // CLEARS the n'th bit
{    int whichByte = n / 8;
     int whichBit = n % 8;
     byteArray [whichByte]  &=  ( (1 << whichBit) ^ 255);
}
```


Constructors for a SET

- The constructor that we use most of the time calculates how many bytes of memory are needed and allocates them using **new**. The set is cleared automatically to all zeroes, yielding the empty set:

```
Bitset (int size)
{  maxSize = size;
   int nbyte = (size + 7) / 8;
   byteArray = new byte[nbyte];
}
Bitset ()
{  maxSize = 0;
   byteArray = null;
}
```

In case we declare an array of sets, we define a 0-arg constructor and an initialization function (named **setSize**). See Bitset.java for details.

How Does Union/Intersect Work?

setA ???????? ???????? ???????? 01101011 01010100

9 6
8 2 13
14
11 4

setB 01100011 01010100

9 6
8 2
14 13
4

setA.union (setB)

setA.intersect (setB)

Defining the Operators

- The union, intersection and difference operations act on 2 sets to produce a third set. What's tricky is making sure these methods work correctly when they operate on sets of different sizes. For example,
  - Bitset union (Bitset setB)

```
{   Bitset temp = new Bitset (maxSize  > setB.maxSize ? this : setB);

    int nbyte = Math.min (byteArray.length, setB.byteArray.length);

    for (int i = 0; i < nbyte; i++)
    {
        temp.byteArray[i] = (byte)  (byteArray[i] | setB.byteArray[i]);
    }
   return temp;
}
```


Other Useful Set Functions

- Frequently we need to operate on a single element of a set: to add it to the set, remove it from the set, or test whether it is present in the set. All these things are easy to do in terms of primitive **Bitset** operations:

```
boolean member  (int i)
{
     if (i >= maxSize)  return false;
     else return (getBit (i) );

}
```

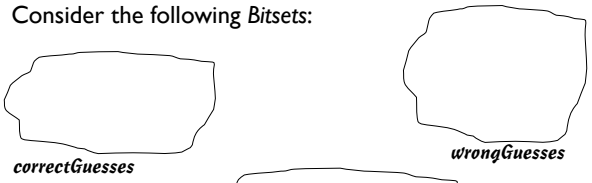
```
void  include (int i)
  {  if  (i >= maxSize) error("Too big!");  setBit(i);  }
```

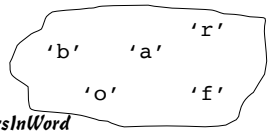
```
void exclude (int i)
  {  if (i  >= maxSize) error ("Too big!");  clearBit(i);  }
```

- All the above check parameter i's validity.

Hangman: A Bitset Application

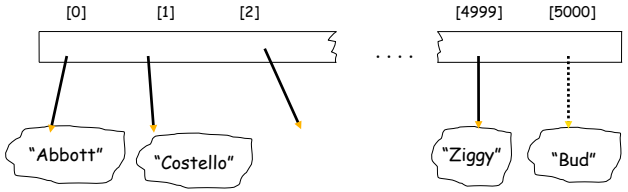
- Consider the following *Bitsets*:





Limitations of Arrays, *revisited*

- Consider employee objects, sorted alphabetically by last name.
  - How do we add / delete objects from the array?



Linked-List Alternative