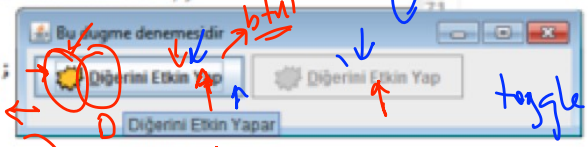
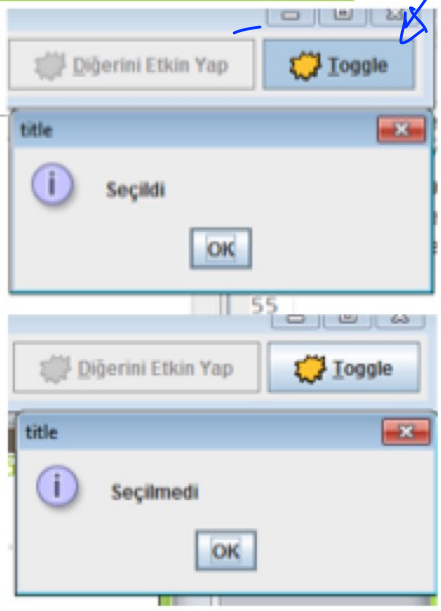
JButton

```
public class Dugmeler extends JFrame implements ActionListener {

    private ImageIcon img;
    private JButton btn;
    private JButton btn2;

    // REsim yüklemek için iç metot
    private ImageIcon ImageIconDondur(String path) {
        java.net.URL imageURL = Dugmeler.class.getResource(path);
        if (imageURL != null) {
            return new ImageIcon(imageURL);
        } else {
            System.err.println("Couldn't find file: " + path);
            return null;
        }

    }

    public Dugmeler() {
        setLayout(new FlowLayout(FlowLayout.LEFT));
        img= ImageIconDondur("images/middle.gif");
        btn = new JButton("Diğerini Etkin Yap",img );
        btn.setMnemonic(KeyEvent.VK_D);
        btn.setActionCommand("btn1");
        btn.setToolTipText("Diğerini Etkin Yapar");
        btn.addActionListener(this);

        btn2 = new JButton("Diğerini Etkin Yap", img);
        btn2.addActionListener(this);
        btn2.setMnemonic(KeyEvent.VK_D);
        btn2.setActionCommand("btn2");
        btn2.setToolTipText("Diğerini Etkin Yapar");
        btn2.setEnabled(false);
        setTitle("Bu dugme denemesidir");
        add(btn);
        add(btn2);
        setSize(200, 200);
        setVisible(true);
    }

    public static void main(String[] args) {

        JFrame gui = new Dugmeler();

    }

    @Override
    public void actionPerformed(ActionEvent e) {

        if ("btn1".equals(e.getActionCommand())) {
            btn.setEnabled(false);
            btn2.setEnabled(true);
            //Enter a basıldığında çalışma
            getRootPane().setDefaultButton(btn2);
        } else if ("btn2".equals(e.getActionCommand())) {
            btn.setEnabled(true);
            btn2.setEnabled(false);
            //Enter a basıldığında çalışma
            getRootPane().setDefaultButton(btn);
        }

    }
}
```



JToggle

```
10      public JButton btn2;
11      public JToggleButton TBtn;
12
42          TBtn = new JToggleButton("Toggle", img);
43          TBtn.addActionListener(this);
44          TBtn.setActionCommand("toggle");
45          TBtn.setMnemonic(KeyEvent.VK_T);

    @Override
    public void actionPerformed(ActionEvent e) {

        if ("btn1".equals(e.getActionCommand())) {
            btn.setEnabled(false);
            btn2.setEnabled(true);
            // Enter a basıldığında çalışma
            getRootPane().setDefaultButton(btn2);
        } else if ("btn2".equals(e.getActionCommand())) {
            btn.setEnabled(true);
            btn2.setEnabled(false);
            // Enter a basıldığında çalışma
            getRootPane().setDefaultButton(btn);
        } else if ("toggle".equals(e.getActionCommand())) {
            if( TBtn.isSelected())
                JOptionPane.showMessageDialog(this, "Seçildi", "title", JOptionPane.INFORMATION_MESSAGE);
            else
                JOptionPane.showMessageDialog(this, "Seçilmedi", "title", JOptionPane.INFORMATION_MESSAGE);
```

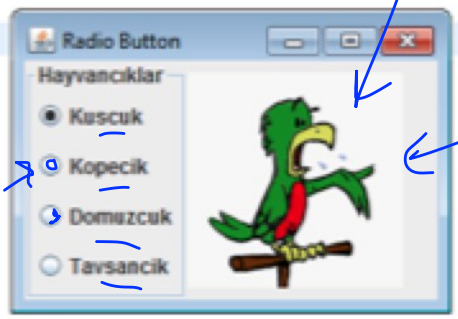


true = isSelected()

false isSelected()

JRadioButton

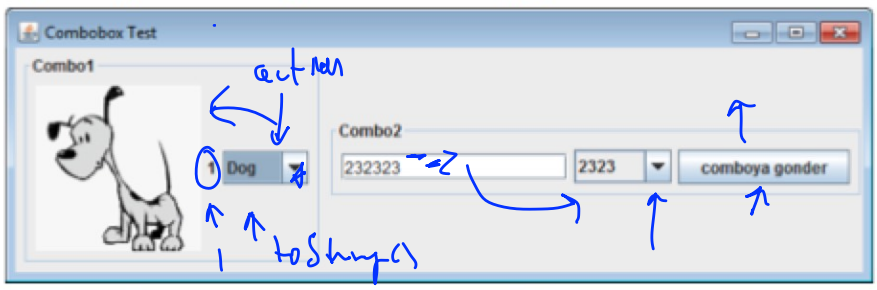
```
public class Checkbox extends JFrame implements ActionListener {
    private JRadioButton rbPig;
    private JRadioButton rbRabbit;
    private JRadioButton rbDog;
    private JRadioButton rbBird;
    private JLabel Resim;

    public Checkbox() {
        setLayout(new GridLayout(1, 2));
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
        setTitle("Radio Button");
        setVisible(true);

        rbPig = new JRadioButton("Domuzcuk");
        rbPig.addActionListener(this);
        rbPig.setActionCommand("images/pig.gif");

        rbRabbit = new JRadioButton("Tavsancik");
        rbRabbit.addActionListener(this);
        rbRabbit.setActionCommand("images/rabbit.gif");
```



JComboBox

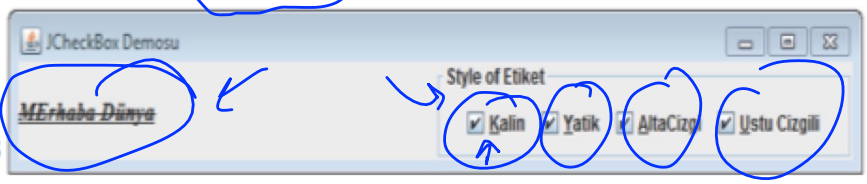
- ComboBox.addActionListener(Obje)
- Obje implements ActionListener
  - ActionPerformed (ActionEvent e)

```
@Override
public void actionPerformed(ActionEvent e) {
    // TODO Auto-generated method stub
    if (e.getSource() == cmbSecim) {
        String str = (String) cmbSecim.getSelectedItem();
        secim.setIcon(REsimYukle(str));
        secim.setText(cmbSecim.getSelectedIndex() + "");
    } else if (e.getSource() == btnSecim) {
        cmbSecim2.addItem(tfSecim.getText());
    } else if (e.getSource() == cmbSecim2) {
            tfSecim.setText((String) cmbSecim2.getSelectedItem());
        }

}
```



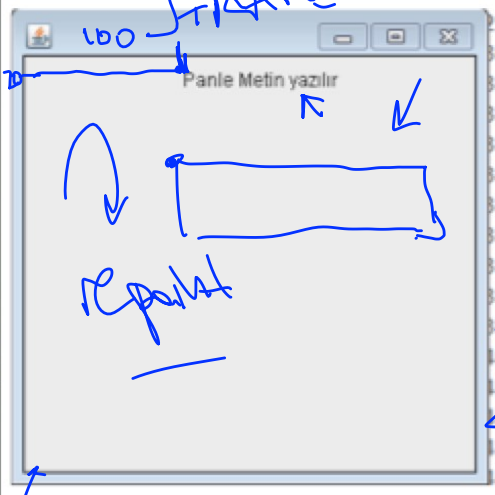
JCheckBox

```
1  import java.awt.*;
2  import java.awt.font.*;
3  import java.awt.event.ItemEvent;
4  import java.awt.event.ItemListener;
5  import java.awt.event.KeyEvent;
6  import java.util.ResourceBundle.Control;
7  import java.util.Map;
8  import javax.swing.*;
9
10 public class theChecjBox extends JFrame implements ItemListener {
11     private JLabel etiket;
12     private JCheckBox Kalin;
13     private JCheckBox Yatik;
14     private JCheckBox AltaCizgi;
15     private JCheckBox UstuCizgili;
16     private Font org;
17
19     public void itemStateChanged(ItemEvent e) {
78
79     public theChecjBox() {
18
19     public static void main(String[] args) {
20         new theChecjBox();
21
22     }
23 }
```



Java ile Paint işlemleri

- Metin Boyama

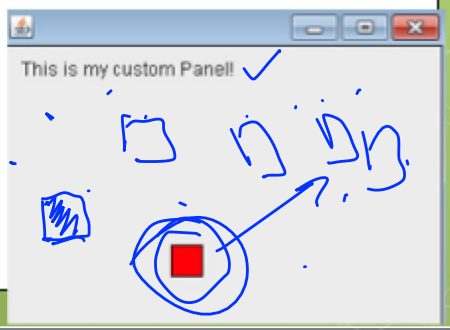



```
29 class MyPanel extends JPanel{
30
31     public MyPanel() {
32         setBorder(BorderFactory.createLineBorder(Color.black));
33     }
34
35     public Dimension getPreferredSize() {
36         return new Dimension(250, 200);
37     }
38
39     public void paintComponent(Graphics g) {
40         super.paintComponent(g);
41
42         // Draw Text
43         g.drawString ("Panle Metin yazılır", 100, 20);
44     }
```

- add(**new MyPanel()**);

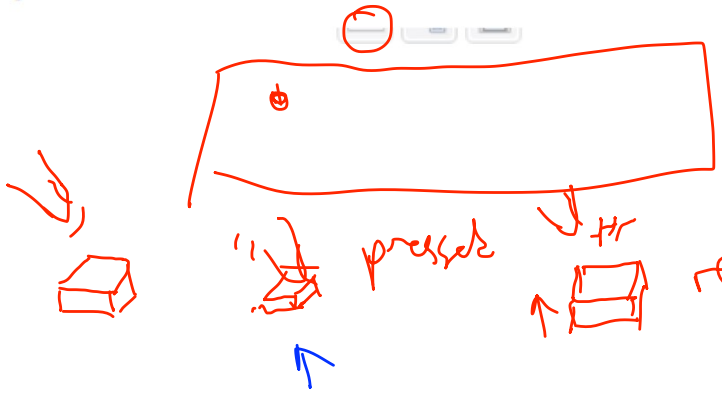
Java ile Boyama MyPanel2

```
class MyPanel extends JPanel {
private int squareX = 50;
private int squareY = 50;
private int squareW = 20;
private int squareH = 20;
public MyPanel2() {

setBorder(BorderFactory.createLineBorder(Color.black));
    addMouseListener(new MouseAdapter() {
       public void mousePressed(MouseEvent e) {
          moveSquare(e.getX(),e.getY());
       }
    });
    addMouseMotionListener(new MouseAdapter() {
       public void mouseDragged(MouseEvent e) {
          moveSquare(e.getX(),e.getY());
       }
    });

  }

  private void moveSquare(int x, int y) {
     int OFFSET = 1;
     if ((squareX!=x) || (squareY!=y)) {

repaint(squareX,squareY,squareW+OFFSET,squareH+O
FFSET);
        squareX=x;
        squareY=y;
repaint(squareX,squareY,squareW+OFFSET,squareH+O
FFSET);
     }
  }

  public Dimension getPreferredSize() {
     return new Dimension(250,200);
  }

  protected void paintComponent(Graphics g) {
     super.paintComponent(g);
     g.drawString("This is my custom Panel!",10,20);
     g.setColor(Color.RED);
     g.fillRect(squareX,squareY,squareW,squareH);
     g.setColor(Color.BLACK);
     g.drawRect(squareX,squareY,squareW,squareH);
  }
}
```



Key Events .. KeyListener

```
5  public class KeyEvents extends JFrame implements KeyListener {
7      public void keyPressed(KeyEvent e) {
11
13     public void keyReleased(KeyEvent e) {
17
19     public void keyTyped(KeyEvent e) {
25      * @param args
27     public static void main(String[] args) {
28         // TODO Auto-generated method stub
29
30     }
31
```

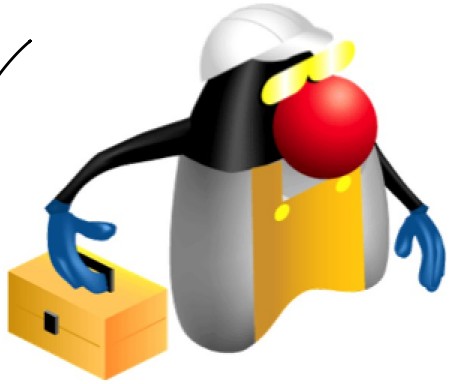


9 Exceptions and Assertions

Error Handling

Applications will encounter errors while executing. Reliable applications should handle errors as gracefully as possible. Errors:

- Should be the "exception" and not the expected behavior
- Must be handled to create reliable applications
- Can occur as the result of application bugs
- Can occur because of factors beyond the control of the application
  - Databases becoming unreachable
  - Hard drives failing



Exception Handling in Java

When using Java libraries that rely on external resources, the compiler will require you to "handle or declare" the exceptions that might occur.

- Handling an exception means you must add in a code block to handle the error.
- Declaring an exception means you declare that a method may fail to execute successfully.

The `try-catch` Statement

The `try-catch` statement is used to handle exceptions.

```
try {
    System.out.println("About to open a file");
    InputStream in =
        new FileInputStream("missingfile.txt");
    System.out.println("File open");
} catch (Exception e) {
    System.out.println("Something went wrong!");
}
```

This line is skipped if the previous line failed to open the file.

This line runs only if something went wrong in the `try` block.

Exception Objects

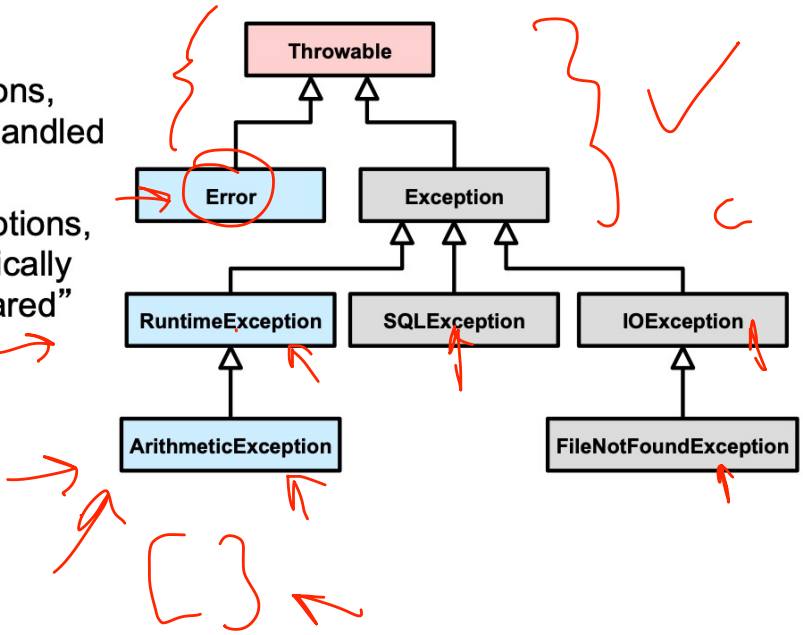
A `catch` clause is passed a reference to a `java.lang.Exception` object. The `java.lang.Throwable` class is the parent class for `Exception` and it outlines several methods that you may use.

```
try{
    //...
} catch (Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}
```

Exception Categories

The `java.lang.Throwable` class forms the basis of the hierarchy of exception classes. There are two main categories of exceptions:

- Checked exceptions, which must be "handled or declared"
- Unchecked exceptions, which are not typically "handled or declared"

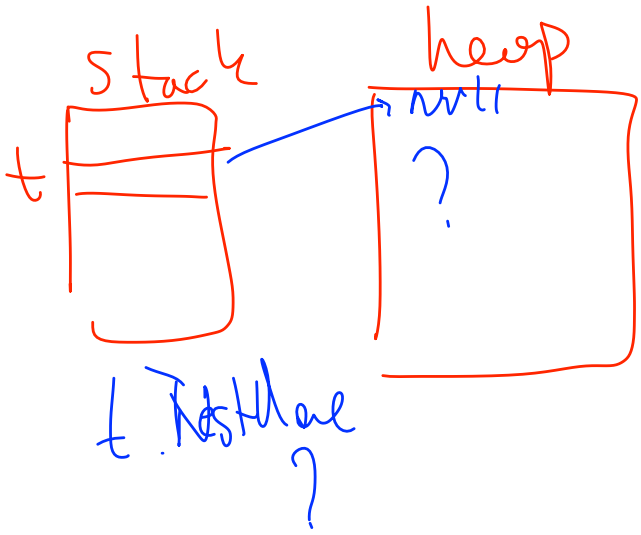



```
1  package myexceptions;
2
3  public class ExceptionExamples {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          Test t=null;
8
9          System.out.println(t.TestName);
10
11         System.out.println("Program ended");
12     }
13
14 }
```

```
Console  Problems  Debug Shell
<terminated> ExceptionExamples [Java Application] /Library/Java/JavaVirtualMachines/jdk-10.0.1.jdk/Contents/Ho
Exception in thread "main" java.lang.NullPointerException
        at Gui/myexceptions.ExceptionExamples.main(ExceptionExamples.java:9)
```

```
MyMenuBar.java  Buttons.java  Combo.java
1  package myexceptions;
2
3  public class ExceptionExamples {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          Test t=null;
8      try {
9          System.out.println(t.TestName);
10     }
11     catch(Exception e) {
12
13     }
14         System.out.println("Program ended");
15     }
16
17 }
```

```
Console  Problems  Debug Shell
<terminated> ExceptionExamples [Java Application] /Library/Java/JavaVi
Program ended
```

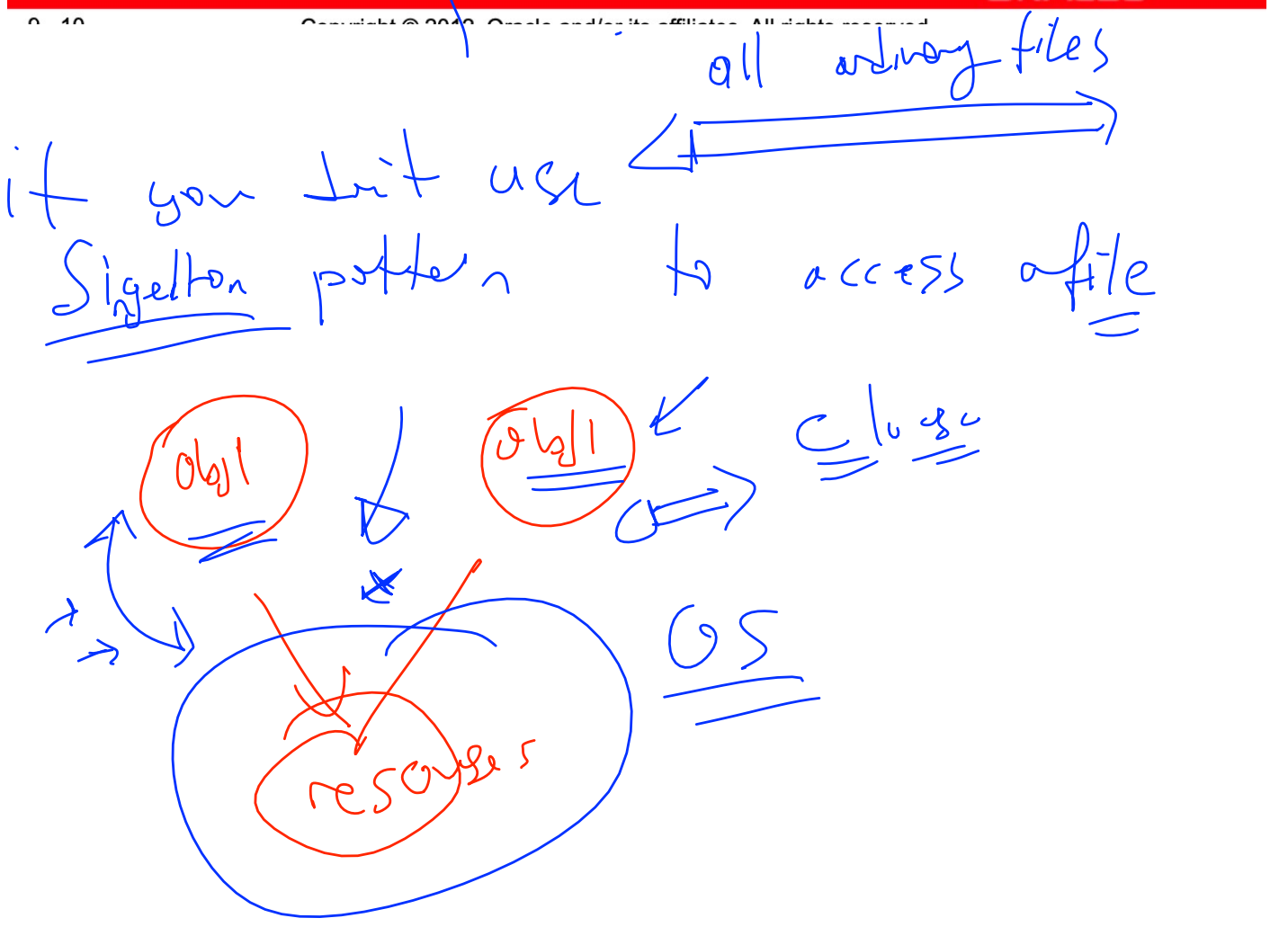


Handling Exceptions

You should always catch the most specific type of exception.
Multiple catch blocks can be associated with a single try.

```
try {
    System.out.println("About to open a file");
    InputStream in = new FileInputStream("missingfile.txt");
    System.out.println("File open");
    int data = in.read();
    in.close();
} catch (FileNotFoundException e) {
    System.out.println(e.getClass().getName());
    System.out.println("Quitting");
} catch (IOException e) {
    System.out.println(e.getClass().getName());
    System.out.println("Quitting");
}
```

Order is important. You must catch the most specific exceptions first (that is, child classes before parent classes).



The `finally` Clause

```
InputStream in = null;
try {
    System.out.println("About to open a file");
    in = new FileInputStream("missingfile.txt");
    System.out.println("File open");
    int data = in.read();
} catch (IOException e) {
    System.out.println(e.getMessage());
} finally {
    try {
        if(in != null) in.close();
    } catch(IOException e) {
        System.out.println("Failed to close file");
    }
}
```

A `finally` clause runs regardless of whether or not an `Exception` was generated.

You always want to close open resources.

The `try`-with-resources Statement

Java SE 7 provides a new `try`-with-resources statement that will autoclose resources.

```
System.out.println("About to open a file");
try (InputStream in =
       new FileInputStream("missingfile.txt")) {
    System.out.println("File open");
    int data = in.read();
} catch (FileNotFoundException e) {
    System.out.println(e.getMessage());
} catch (IOException e) {
    System.out.println(e.getMessage());
}
```

Suppressed Exceptions

If an exception occurs in the `try` block of a `try`-with-resources statement *and* an exception occurs while closing the resources, the resulting exceptions will be suppressed.

```
} catch(Exception e) {
    System.out.println(e.getMessage());
    for(Throwable t : e.getSuppressed()) {
        System.out.println(t.getMessage());
    }
}
```

The `AutoCloseable` Interface

Resource in a `try`-with-resources statement must implement either:

- `java.lang.AutoCloseable`
  - New in JDK 7
  - May throw an `Exception`
- `java.io.Closeable`
  - Refactored in JDK7 to extend `AutoCloseable`
  - May throw an `IOException`

```
public interface AutoCloseable {
    void close() throws Exception;
}
```

Catching Multiple Exceptions

Java SE 7 provides a new multi-`catch` clause.

```
ShoppingCart cart = null;
try (InputStream is = new FileInputStream(cartFile);
     ObjectInputStream in = new ObjectInputStream(is)) {
      cart = (ShoppingCart)in.readObject();
} catch (ClassNotFoundException | IOException e) {
    System.out.println("Exception deserializing " + cartFile);
    System.out.println(e);
    System.exit(-1);
}
```

Multiple exception types are separated with a vertical bar.

Declaring Exceptions

You may declare that a method throws an exception instead of handling it.

```
public static int readByteFromFile() throws IOException {
    try (InputStream in = new FileInputStream("a.txt")) {
        System.out.println("File open");
        return in.read();
    }
}
```

Notice the lack of `catch` clauses. The `try`-with-resources statement is being used only to close resources.

Handling Declared Exceptions

The exceptions that methods may throw must still be handled. Declaring an exception just makes it someone else's job to handle them.

```
public static void main(String[] args) {
    try {
        int data = readByteFromFile();
    } catch (IOException e) {
        System.out.println(e.getMessage());
    }
}
```

Method that declared an exception

Throwing Exceptions

You can rethrow an exception that has already been caught. Note that there is both a `throws` clause and a `throw` statement.

```
public static int readByteFromFile() throws IOException {
    try (InputStream in = new FileInputStream("a.txt")) {
        System.out.println("File open");
        return in.read();
    } catch (IOException e) {
        e.printStackTrace();
        throw e;
    }
}
```

Custom Exceptions

You can create custom exception classes by extending `Exception` or one of its subclasses.

```
public class DAOException extends Exception {

    public DAOException() {
        super();
    }

    public DAOException(String message) {
        super(message);
    }
}
```


Wrapper Exceptions

To hide the type of exception being generated without simply swallowing the exception, use a wrapper exception.

```
public class DAOException extends Exception {
    public DAOException(Throwable cause) {
        super(cause);
    }

    public DAOException(String message, Throwable cause)
    {
        super(message, cause);
    }
}
```

Decrease Font Size

Revisiting the DAO Pattern

Data Access Object

The DAO pattern uses abstraction (an interface) to allow implementation substitution. A file or database DAO must deal with exceptions. A DAO implementation may use a wrapper exception to preserve the abstraction and avoid swallowing exceptions.

```
public Employee findById(int id) throws DAOException {
    try {
        return employeeArray[id];
    } catch (ArrayIndexOutOfBoundsException e) {
     throw new DAOException("Error finding employee in DAO", e);
    }
}
```

Wrapper Exceptions

To hide the type of exception being generated without simply swallowing the exception, use a wrapper exception.

```
public class DAOException extends Exception {
    public DAOException(Throwable cause) {
        super(cause);
    }

    public DAOException(String message, Throwable cause)
    {
        super(message, cause);
    }
}
```

Assertions

- Use assertions to document and verify the assumptions and internal logic of a single method:
  - Internal invariants
  - Control flow invariants
  - Postconditions and class invariants
- Inappropriate uses of assertions

  Assertions can be disabled at run time; therefore:
  - Do not use assertions to check the parameters of a public method.
  - Do not use methods that can cause side effects in the assertion check.

Assertion Syntax

- Syntax of an assertion is:

```
assert <boolean_expression> ;
assert <boolean_expression> : <detail_expression> ;
```

- If *<boolean_expression>* evaluates *false*, then an `AssertionError` is thrown.
- The second argument is converted to a string and used as descriptive text in the `AssertionError` message.

Internal Invariants

- The problem is:

```
1   if (x > 0) {
2     // do this
3   } else {
4     // do that
5   }
```

- The solution is:

```
1   if (x > 0) {
2     // do this
3   } else {
4     assert ( x == 0 );
5     // do that, unless x is negative
6   }
```

Control Flow Invariants

Example:

```
1  switch (suit) {
2      case Suit.CLUBS: // ...
3        break;
4      case Suit.DIAMONDS: // ...
5        break;
6      case Suit.HEARTS: // ...
7        break;
8      case Suit.SPADES: // ...
9        break;
10     default: assert false : "Unknown playing card suit";
11        break;
12  }
```

Postconditions and Class Invariants

Example:

```
1   public Object pop() {
2     int size = this.getElementCount();
3     if (size == 0) {
4       throw new RuntimeException("Attempt to pop from empty stack");
5     }
6
7     Object result = /* code to retrieve the popped element */ ;
8
9     // test the postcondition
10      assert (this.getElementCount() == size - 1);
11
12    return result;
13    }
```

Controlling Runtime Evaluation of Assertions

- If assertion checking is disabled, the code runs as fast as if the check were never there.
- Assertion checks are disabled by default. Enable assertions with either of the following commands:

```
java -enableassertions MyProgram
```

```
java -ea MyProgram
```

- Assertion checking can be controlled on class, package, and package hierarchy basis. See: http://download.oracle.com/javase/7/docs/technotes/guides/language/assert.html