|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| java.net.Socket; ObjectInputStream; ObjectOutputStream;BufferedReader; InputStreamReader;  public class EchoClient {  public static void main(String[] args) {  try { String serverName;  System.out.println("Enter Server:");  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));  serverName = in.readLine();  Socket sock = new Socket(serverName, EchoServer.SERVER\_PORT);  ObjectOutputStream output = new ObjectOutputStream(sock.getOutputStream());  ObjectInputStream input = new ObjectInputStream(sock.getInputStream());  Message msg = null, resp = null;  do { msg = new Message(readText());  output.writeObject(msg); //get ack  resp = (Message) input.readObject();  System.out.println("\nServer says: " + resp.theMessage + "\n");  } while (!msg.theMessage.toUpperCase().equals("EXIT")); sock.close(); } }  private static String readText() {  try { System.out.println("Enter text");  System.out.print(" > ");  BufferedReader in = new BufferedReader(new InputStreamReader(System.in));  return in.readLine(); } } } | | ; java.lang.Thread; java.net.Socket;ObjectInputStream; java.io.ObjectOutputStream;  public class EchoThread extends Thread  { private final Socket socket;  public EchoThread(Socket \_socket)  { socket = \_socket; }  public void run() { try { ObjectInputStream input = new ObjectInputStream(socket.getInputStream());  ObjectOutputStream output = new ObjectOutputStream(socket.getOutputStream());  Message msg = null;  int count = 0; do{msg = (Message)input.readObject();  System.out.println("[" + socket.getInetAddress() + ":" + socket.getPort() + "] " + msg.theMessage); //ack  count++;  }while(!msg.theMessage.equals("exit"));  socket.close(); } } }  public class Message implements java.io.Serializable  { public String theMessage;  public Message(String \_msg){  theMessage = \_msg; } } | | | |
| java.net.ServerSocket; java.net.Socket;  public class EchoServer  { public static final int SERVER\_PORT = 8765;  public static void main(String[] args)  {try{  final ServerSocket serverSock = new ServerSocket(SERVER\_PORT);  Socket sock = null;  EchoThread thread = null;  while(true){ sock = serverSock.accept();  thread = new EchoThread(sock);  thread.start(); } }  catch(Exception e){  System.err.println("Error: " + e.getMessage());  e.printStackTrace(System.err); } } } | | import java.util.ArrayList; import java.util.Collections; public class TestStudent { public static void main(String[] args) {  ArrayList<Student> studentsList = new ArrayList<Student>(); Student s = new Student("Alice", 3.5); studentsList.add(s); s = new Student("John", 2.35); studentsList.add(s); s = new Student("Peter", 3.1);  studentsList.add(s); Collections.sort(studentsList);  for(Student ss: studentsList)  System.out.println("Name: " + ss.getName() + " , GPA: " + ss.getGpa()); } }  package comparable;  public class Student implements Comparable { private String name; private double gpa;  public Student(String name, double gpa) {  this.name = name; this.gpa = gpa; }  public int compareTo(Object o) { if (((Student) o).gpa < gpa) { return 1;  } else if (((Student) o).gpa > gpa) { return -1;  } else { return 0; } } public double getGpa() { return gpa; }  public String getName() { return name; } } | | | |
| package testsequence;  public class TestSequence {  public static void main(String[] args) {  Sequence s = new Sequence(10);  for (int i = 0; i < 10; i++) {  s.add(Integer.toString(i)); }  Selector sl = s.getSelector();  while (!sl.end()) {  System.out.println(sl.current());  sl.next(); } } } | public class Sequence { private Object[] objects; private int next = 0; public Sequence(int size) { objects = new Object[size]; }  public void add(Object x) { if (next < objects.length) {  objects[next] = x;  next++; } }// inner  private class SSelector implements Selector { int i = 0;  public boolean end() {return i == objects.length;}  public Object current() {return objects[i]; }  public void next() {if (i < objects.length) {  i++; } } } //inner class ends  public Selector getSelector() { return new SSelector(); } } | | | package testsequence;  public interface Selector {  boolean end();  Object current();  void next();} | |
| package comparable;  import java.util.ArrayList;  import java.util.Collections;  public class TestStudent {  public static void main(String[] args) {  ArrayList<Student> studentsList = new ArrayList<Student>();  Student s = new Student("Alice", 3.5);  studentsList.add(s);  s = new Student("John", 2.35);  studentsList.add(s);  s = new Student("Peter", 3.1);  studentsList.add(s);  Collections.sort(studentsList);  for(Student ss: studentsList)  System.out.println("Name: " + ss.getName() + " , GPA: " + ss.getGpa()); } } | | | package comparable;  public class Student implements Comparable {  private String name;  private double gpa;  public Student(String name, double gpa) {  this.name = name;  this.gpa = gpa; }  public int compareTo(Object o) {  if (((Student) o).gpa < gpa) {  return 1;  } else if (((Student) o).gpa > gpa) { return -1;  } else { return 0; } }  public double getGpa() {  return gpa; }  public String getName() {  return name; } } | |
| public class Salary extends Employee  { private double salary; //Annual salary  public Salary(String name, String address, int number, double  salary) {  super(name, address, number);  setSalary(salary); }  public void mailCheck()  { System.out.println("Within mailCheck of Salary class ");  System.out.println("Mailing check to " + getName()  + " with salary " + salary); }  public double getSalary()  { return salary; }  public void setSalary(double newSalary)  { if(newSalary >= 0.0)  { salary = newSalary; }  }  public double computePay() | | | * + Integer anIntegerObject = new Integer(25);   + int I = anIntegerObject.intValue(); * public final class FinalPersonClass{   **Interface** - Extreme abstract class. Designed to specify methods the concrete class will use. It is not a class. Interface can have no data members. Multiple Inheritance from base classes not allowed. Can only be public. **Implements** -a concrete class must include Implements at start of class def. Must implement all method headings.  **Extends** -may be derived from a base interface using Ex tends. Concrete class that implements a derived interface must have definitions for any methods in the derived interface as well as any methods in the base interface  **Inner Class** - Can reference private variable/method in outer.Outer can reference inner private variable, or class if inner class is calling object.  **Polymorphism** is the ability to associate many meanings to one method name. Late/Dyn **Late binding** for all methods (except private, final, and static methods) **List<T>**:  Provides ordered and indexed collection which may contain duplicates. Ordered, Duplicates, Positional, Null Duplicates. =Arraylist, LinkedList, Vector **Set<T>**:  A collection that contains no duplicate elements. Unordered, No Duplicates, NoPosition, 1 Null, Hashset, LinkedHashset, Treeset. **ArrayList-**   AL is newer and faster.ArrayList is implemented using an array as a private instance variable. When full moves to new array, autogrow but not shrink. Only Class. Not primitive. ArrayList list.add("something");  //Add is overloaded. int howMany = list.size();   //find size  list.set(index, "something else");  String thing = list.get(index); **Abstract Method** - placeholder. Can’t be private, complete heading, with no body. Abstract any number of methods. can have abstracts derived classes.Abs Class Const can’t be used to create an obj of abs class. Super is passed as abs **Information hiding/Abstraction** is the practice of separating how to use a class from the details of its implementation **Encapsulation** means that the data and methods of a class are combined into a single unit (i.e., a class object), which hides the implementation details. Done by making them private. **Inheritance** is the process by which a new class is created from another class  The new class is called a derived class  The original class is called the base class | |

|  |  |  |  |
| --- | --- | --- | --- |
| package arrayslistdemo;  import java.util.ArrayList;  class AABattery { @Override  public String toString() {  return "AABattery"; }}  class NineVoltBattery {  @Override  public String toString() {  return "NineVoltBattery"; } }  class RollOfRibbon {  @Override  public String toString() {  return "RollOfRibbon"; }}  class PaperClip {  int i;  PaperClip(int i) {  this.i = i; }  @Override  public String toString() {  return "PaperClip(" + i + ")"; }}  class BandAid {  @Override  public String toString() {  return "BandAid"; } }  class Box {  ArrayList moreStuff = new ArrayList();  @Override  public String toString() {  String s = new String("Box");  s += moreStuff;  return s; }}  class BoxOfPaperClips {  ArrayList clips = new ArrayList();  @Override  public String toString() {  String s = new String("BoxOfPaperClips");  s += clips;  return s; }}  class JunkDrawer {  ArrayList contents = new ArrayList();  public void fillDrawer() {  contents.add(new RollOfRibbon());  contents.add(new AABattery());  contents.add(new NineVoltBattery());  BoxOfPaperClips boxOfClips = new BoxOfPaperClips();  for (int i = 0; i < 3; i++) {  boxOfClips.clips.add(new PaperClip(i)); }  contents.add(boxOfClips); //Note we're adding an array reference inside another array  Box box = new Box();  box.moreStuff.add(new AABattery());  box.moreStuff.add(new BandAid());  contents.add(box);  contents.add(new AABattery()); }}  public class ArraysListDemo {  public static void main(String[] args) {  JunkDrawer kitchenDrawer = new JunkDrawer();  kitchenDrawer.fillDrawer();  System.out.println(kitchenDrawer.contents); }}  public class BufferedReaderDemo {  public static void main(String[] args) {  try {  //This code illustrates how to open a text file to read data from line by line.  //1- Intialize a BufferedReader object that will point to a reference of a FileReader object holding your filename  // Substitue "jacksong.txt" with your file name  BufferedReader inputStream =  new BufferedReader(new FileReader("jacksong.txt"));  String line;  //2- Loop over the file using BufferedReader object and the readLine() method, which will read  // a single line in the text file at a time and return it in a String object  while ((line = inputStream.readLine()) != null) {  //3- Put your line processing code here - this code just prints out the line read from file  System.out.println(line);  }  //4- Close your BufferedReader stream  inputStream.close();  } catch (FileNotFoundException e) {  System.out.println("File jacksong.txt was not found");  System.out.println("or could not be opened.");  } catch (IOException e) {  System.out.println("Error reading from morestuff2.txt."); } }} | | package discountsale;  public class Sale {  private String name; //A nonempty string  private double price; //nonnegative  public Sale() {  name = "No name yet";  price = 0; }  public Sale(String theName, double thePrice) {  setName(theName);  setPrice(thePrice); }  public Sale(Sale originalObject) {  if (originalObject == null) { System.out.println("Error: null Sale object.");  System.exit(0); }  name = originalObject.name;  price = originalObject.price; }  public static void announcement() {  System.out.println("This is the Sale class."); }  public double getPrice() {  return price; }  public void setPrice(double newPrice) {  if (newPrice >= 0) {  price = newPrice;  } else {  System.out.println("Error: Negative price.");  System.exit(0); } }  public String getName() {  return name; } public void setName(String newName) {  if (newName != null && newName != "") {  name = newName;  } else {  System.out.println("Error: Improper name value.");  System.exit(0);  } }  public String toString() {  return (name + " Price and total cost = $" + price); }  public double bill() {  return price; }  public boolean equalDeals(Sale otherSale) {  if (otherSale == null) {  return false;  } else {  return (name.equals(otherSale.name)  && bill() == otherSale.bill()); } }  public boolean lessThan(Sale otherSale) {  if (otherSale == null) {  System.out.println("Error: null Sale object.");  System.exit(0); } //else  return (bill() < otherSale.bill()); }  public boolean equals(Object otherObject) {  if (otherObject == null) {  return false;  } else if (getClass() != otherObject.getClass()) {  return false;  } else {  Sale otherSale = (Sale) otherObject;  return (name.equals(otherSale.name)  && (price == otherSale.price)); } }  public Sale clone() {  return new Sale(this); }}  public class LateBindingDemo  {  public static void main(String[] args)  {  Sale simple = new Sale("floor mat", 10.00);  DiscountSale discount = new DiscountSale("floor mat", 11.00, 10);  System.out.println(simple);  System.out.println(discount);  if (discount.lessThan(simple))  System.out.println("Discounted item is cheaper.");  else  System.out.println("Discounted item is not cheaper.");  Sale regularPrice = new Sale("cup holder", 9.90);//One item at $9.90.  DiscountSale specialPrice = new DiscountSale("cup holder", 11.00, 10);  System.out.println(regularPrice);  System.out.println(specialPrice);  if (specialPrice.equalDeals(regularPrice))  System.out.println("Deals are equal.");  else  System.out.println("Deals are not equal.");  }  } | |
| package assig2;  public class RCondition implements Runnable {  private int count = 0;  private int increment = 1;  public void run() {  increment();  System.out.println(count); }  public void increment(){  int local = count;  local += increment;  count = local; } }  package threadsdemo;  public class ThreadTester {  public static void main(String[] args) {  Runnable r1 = new GreetingProducer("Hello");  Runnable r2 = new GreetingProducer("Goodbye");  Thread t1 = new Thread(r1);  Thread t2 = new Thread(r2);  t1.start(); t2.start(); } }  package threadsdemo;  class ImplementsRunnable implements Runnable {  private int counter = 0;  public void run() { | counter++;  System.out.println("ImplementsRunnable:counter " + counter); }}  class ExtendsThread extends Thread {  private int counter = 0;  public void run() { counter++;  System.out.println("ExtendsThread : Counter : " + counter); }}  public class ThreadVsRunnable {  public static void main(String args[]) throws Exception {  ImplementsRunnable rc = new ImplementsRunnable();  Thread t1 = new Thread(rc);  t1.start();  Thread.sleep(1000);  Thread t2 = new Thread(rc);  t2.start();  Thread.sleep(1000);  Thread t3 = new Thread(rc);  t3.start();  ExtendsThread tc1 = new ExtendsThread();  tc1.start();  Thread.sleep(1000); | ExtendsThread tc2 = new ExtendsThread();  tc2.start();  Thread.sleep(1000);  ExtendsThread tc3 = new ExtendsThread();  tc3.start(); }}  package threadsdemo;  public class GreetingProducer implements Runnable {  private String greeting;  private static final int REPETITIONS = 10;  private static final int DELAY = 100;  public GreetingProducer(String aGreeting) {  greeting = aGreeting; }  public void run() { try {  for (int i = 1; i <= REPETITIONS; i++) {  System.out.println(i + ": " + greeting);  Thread.sleep(DELAY); }  } } } |  |