// UploadServlet  
import java.io.\*;  
import jakarta.servlet.\*.  
  
@MultipartConfig  
public class UploadServlet extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response)  
 throws ServletException, IOException {  
 String dirList = getListing("C:\\tomcat\\webapps\\upload\\images");  
 response.setContentType("text/plain");  
 response.setContentLength(dirList.length());  
 PrintWriter out = response.getWriter();  
 out.println(dirList);   
 out.flush();   
 }   
 @Override  
 protected void doPost(HttpServletRequest request, HttpServletResponse response)  
 throws ServletException, IOException {  
System.*out*.println("??????? in do Post??????\n");  
 Part filePart = request.getPart("File");  
 String fileName = filePart.getSubmittedFileName();  
 if(fileName.equals("")){  
 response.setStatus(302);  
 return;  
 }  
System.*out*.println("??????? inside do Post??????\n");  
 filePart.write(System.*getProperty*("catalina.base") + "/webapps/upload/images/" + fileName);  
   
 }   
 private String getListing(String path) {  
 System.*out*.println("??????? in getListing??????\n");  
 String dirList = null;  
 File dir = new File(path);  
 String[] chld = dir.list();  
 for(int i = 0; i < chld.length; i++){  
 dirList += "," + chld[i];   
 }  
 System.*out*.println(dirList);  
 return dirList;  
 }   
}

// index2.html  
<form action="/upload/upload" method= "POST" enctype="multipart/form-data">  
 Select a file: <input type="file" name="File"><br><br>  
 <input type="submit">  
</form>

// web.xml  
<web-app> xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" version="3.0">  
 <servlet>  
 <servlet-name>bob</servlet-name>  
 <servlet-class>UploadServlet</servlet-class>  
 </servlet>  
 <servlet-mapping>  
 <servlet-name>bob</servlet-name>  
 <url-pattern>/upload</url-pattern>  
 </servlet-mapping>  
</web-app>

// Trivia

// FileUploadServlet  
import jakarta.servlet.\*;   
import java.sql.\*;  
import java.time.LocalDate;  
import java.io.\*;  
import java.lang.StringBuilder;  
import java.util.\*;  
import java.text.\*;  
import java.nio.\*;  
  
@MultipartConfig  
public class FileUploadServlet extends HttpServlet {  
 @Override  
 protected void doGet(HttpServletRequest request, HttpServletResponse response)  
 throws ServletException, IOException {  
 response.setContentType("text/html");  
 response.setCharacterEncoding("UTF-8");  
 PrintWriter out = response.getWriter();  
 out.println(  
"<!DOCTYPE html>" +  
"<head>" +  
"<title>File Upload Form</title>" +  
"</head>" +  
"<body>" +  
"<h1>Upload file</h1>" +  
"<form method=\"POST\" action=\"upload\"" +  
"enctype=\"multipart/form-data\">" +  
"<input type=\"file\" name=\"FileName\"/>" +  
"Caption: <input type=\"text\" name=\"Question\"/>" +  
"Date: <input type=\"date\" name=\"UploadDate\"/>" +  
"<input type=\"submit\" value=\"Submit\"/>" +  
"</form>" +  
"</body>" +  
"</html>"  
 );  
 }  
  
 @Override  
 protected void doPost(HttpServletRequest request, HttpServletResponse response)  
 throws ServletException, IOException {  
 Part filePart = request.getPart("FileName");  
 String question = request.getParameter("Question");  
 String uploadDate = request.getParameter("UploadDate");  
 String fileName = filePart.getSubmittedFileName();  
 if(uploadDate.equals("")) uploadDate = "2020-10-10";  
 if(question.equals("")) question = "No Question";  
 System.*out*.println(">>>>>" + question + uploadDate + fileName);  
 Connection con = null;  
 try { Class.*forName*("oracle.jdbc.OracleDriver"); } catch (Exception ex) {  
 System.*out*.println("Message: " + ex.getMessage ());   
 return;  
 }  
 try {  
 con = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:XE", "system", "oracle1");  
 PreparedStatement preparedStatement = con.prepareStatement("INSERT INTO trivias (ID,Question, ContentPath, Content) VALUES (?,?,?,?)");  
 UUID uuid = UUID.*randomUUID*();  
 preparedStatement.setBytes(1,*asBytes*(uuid));  
 preparedStatement.setString(2, question);  
 preparedStatement.setString(3, "image/jpeg");  
 preparedStatement.setBinaryStream(4, filePart.getInputStream());  
 int row = preparedStatement.executeUpdate();  
 preparedStatement.close();  
 //con.close();  
 } catch(SQLException ex) {  
 while (ex != null) {   
 System.*out*.println("Message: " + ex.getMessage ());   
 System.*out*.println("SQLState: " + ex.getSQLState ());   
 System.*out*.println("ErrorCode: " + ex.getErrorCode ());   
 ex = ex.getNextException();   
 System.*out*.println("");  
 }   
 }  
/\*  
<img src="  
f3//Ub//ge8WSLf/rhf/3kdbW1mxsbP//mf///yH5BAAAAAAALAAAAAAQAA4AAARe8L1Ekyky67  
QZ1hLnjM5UUde0ECwLJoExKcppV0aCcGCmTIHEIUEqjgaORCMxIC6e0CcguWw6aFjsVMkkIr7g7  
7ZKPJjPZqIyd7sJAgVGoEGv2xsBxqNgYPj/gAwXEQA7"  
 width="16" height="14">  
\*/  
 byte bArr[] = null;  
 UUID sid = null;  
 try {  
 Statement stmt2 = con.createStatement( );  
 ResultSet rs = stmt2.executeQuery("SELECT id, question, contentpath, content FROM trivias");  
 rs.next();  
 byte[] raw = rs.getBytes(1);  
 sid = *asUuid*(raw);  
 question = rs.getString(2);  
 String contentPath = rs.getString(3);  
 Blob b = rs.getBlob(4);  
 bArr = b.getBytes(1, (int) b.length());  
 stmt2.close();  
 con.close();  
 } catch(SQLException ex) {  
 while (ex != null) {   
 System.*out*.println("Message: " + ex.getMessage ());   
 System.*out*.println("SQLState: " + ex.getSQLState ());   
 System.*out*.println("ErrorCode: " + ex.getErrorCode ());   
 ex = ex.getNextException();   
 System.*out*.println("");  
 }   
 }  
 response.setContentType("text/html");  
 PrintWriter out = response.getWriter();  
 String page = "<!DOCTYPE html><html><body>" +  
 "<img src=\"data:image/jpeg;base64," +  
 Base64.*getEncoder*().encodeToString(bArr) + "\"" +  
 " width=\"500\" height=\"500\"></img>" +  
 "</body></html>";  
System.*out*.println(page);  
 out.println(page);   
 }  
 public static byte[] asBytes(UUID uuid) {  
 ByteBuffer bb = ByteBuffer.*wrap*(new byte[16]);  
 bb.putLong(uuid.getMostSignificantBits());  
 bb.putLong(uuid.getLeastSignificantBits());  
 return bb.array();  
 }  
 public static UUID asUuid(byte[] bytes) {  
 ByteBuffer bb = ByteBuffer.*wrap*(bytes);  
 long firstLong = bb.getLong();  
 long secondLong = bb.getLong();  
 return new UUID(firstLong, secondLong);  
 }  
}

// LoginServlet  
import jakarta.servlet.\*;  
import java.sql.\*;  
import java.io.\*;  
public class LoginServlet extends HttpServlet {  
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 response.setContentType("text/html");  
 PrintWriter out = response.getWriter();  
 out.println("<html>\n" + "<head><title>" + "Login" + "</title></head>\n" + "<body>\n"  
 + "<h1 align=\"center\">" + "Login" + "</h1>\n" + "<form action=\"login\" method=\"POST\">\n"  
 + "Username: <input type=\"text\" name=\"user\_id\">\n" + "<br />\n"  
 + "Password: <input type=\"password\" name=\"password\" />\n" + "<br />\n"  
 + "<input type=\"submit\" value=\"Sign in\" />\n" + "</form>\n"  
 + "</form>\n" + "</body>\n</html\n");  
   
 }  
  
 public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 response.setContentType("text/html");  
 String errMsg = "";  
 Connection con = null;  
 try {  
 try { Class.*forName*("oracle.jdbc.OracleDriver"); } catch (Exception ex) { }  
 con = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:XE", "system", "oracle1");  
 Statement stmt2 = con.createStatement();  
 ResultSet rs = stmt2.executeQuery("select \* from accounts");  
 while (rs.next()) {  
 String username = rs.getString("username");  
 String password = rs.getString("password");  
 System.*out*.println(" " + username + " " + password);   
 }  
 stmt2.close();  
 con.close();  
 System.*out*.println("\n\n");  
 } catch(SQLException ex) {   
 errMsg = errMsg + "\n--- SQLException caught ---\n";   
 while (ex != null) {   
 errMsg += "Message: " + ex.getMessage ();   
 errMsg += "SQLState: " + ex.getSQLState ();   
 errMsg += "ErrorCode: " + ex.getErrorCode ();   
 ex = ex.getNextException();   
 errMsg += "";  
 }   
 }   
 PrintWriter out = response.getWriter();  
 response.setContentType("text/html");  
  
 String title = "Logged in as: ";  
 String username = request.getParameter("user\_id");  
 String password = request.getParameter("password");  
 HttpSession session = request.getSession(true);  
 session.setAttribute("USER\_ID", username);  
 response.setStatus(302);  
 response.sendRedirect("main");   
 }  
}

// LogoutServlet  
import java.io.IOException;  
import jakarta.servlet.\*;  
  
public class LogoutServlet extends HttpServlet {  
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 // Set response content type  
  
 HttpSession session = request.getSession(false);  
 if (session != null && request.isRequestedSessionIdValid()) {  
 session.invalidate();  
 }  
 response.sendRedirect("login");  
 }  
// Method to handle POST method request.  
 public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 // Set response content type  
 HttpSession session = request.getSession(false);  
 if (session != null && request.isRequestedSessionIdValid()) {  
 session.invalidate();  
 }  
 response.sendRedirect("login");  
 }  
}

// MainServlet  
import jakarta.servlet.\*;  
import java.io.\*;  
  
public class MainServlet extends HttpServlet {  
 public void doGet(HttpServletRequest request,  
 HttpServletResponse response)  
 throws ServletException, IOException {  
 HttpSession session = request.getSession(false);  
 if (session == null) {  
 response.setStatus(302);  
 response.sendRedirect("login");   
 }   
 String title = "Logged in as: ";  
 title += session.getAttribute("USER\_ID");  
 response.setContentType("text/html");  
 String docType = "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";  
 String html = docType + "<html>\n" + "<head><title>" + title + "</title></head>\n"  
 + "<body bgcolor=\"#f0f0f0\">\n" + "<h1 align=\"center\">" + title + "</h1>\n"+  
 "<div style=\"text-align: center;\">\n" +  
 "<form action=\"upload\" method=\"GET\">\n" +   
 "<input type=\"submit\" value=\"UPLOAD\" />\n" +  
 "</form>\n" +  
 "</div>\n" +  
 "<div style=\"text-align: center;\">\n" +  
 "<form action=\"play\" method=\"GET\">\n" +   
 "<input type=\"submit\" value=\"GALLERY\" />\n" +  
 "</form>\n" +  
 "</div>\n" +  
 "<div style=\"text-align: center;\">\n" +  
 "<form action=\"logout\" method=\"GET\">\n" +   
 "<input type=\"submit\" value=\"LOGOUT\" />\n" +  
 "</form>\n" +  
 "</div>\n" + "</body></html>";  
  
 PrintWriter out = response.getWriter();  
 out.println(html);  
 }  
}

// PlayServlet  
import jakarta.servlet.\*;  
import jakarta.servlet.http.\*;  
import java.sql.\*;  
import java.io.\*;  
public class PlayServlet extends HttpServlet {  
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 String errMsg = "";  
 Connection con = null;  
 try {  
 try { Class.*forName*("oracle.jdbc.OracleDriver"); } catch (Exception ex) { }  
 con = DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:1521:XE", "system", "oracle1");  
 Statement stmt = con.createStatement();  
 ResultSet rs = stmt.executeQuery("select id, question, contentpath from trivias");  
 while (rs.next()) {  
 String content = rs.getString("contentpath");  
   
 }  
 stmt.close();  
 con.close();  
 } catch(SQLException ex) {   
 errMsg = errMsg + "\n--- SQLException caught ---\n";   
 while (ex != null) {   
 errMsg += "Message: " + ex.getMessage ();   
 errMsg += "SQLState: " + ex.getSQLState ();   
 errMsg += "ErrorCode: " + ex.getErrorCode ();   
 ex = ex.getNextException();   
 errMsg += "";  
 }   
 }   
 String contentPath = "tgbNymZ7vqY";  
 response.setContentType("text/html");  
 PrintWriter out = response.getWriter();  
 out.println(   
"<!DOCTYPE html>" +  
"<meta charset='UTF-8'>" +  
"<body>" +  
"<div>" +  
"<iframe id=\"Video\" width=\"420\" height=\"345\" src=https://www.youtube.com/embed/" + contentPath +"?autoplay=1&mute=1&start=62&end=162>" +  
"</iframe>" +  
"</div>" +  
"<div>" +  
"<form action='/trivia/play' method='GET'>" +  
"<br>" +  
"<div class='button'>" +   
"<button class='button' id='prev'>Prev</button>" +  
"<button class='button' id='next'>Next</button>" +  
"</div>" +  
"<br>" +  
"</form>" +  
"<div>" +  
"<form action='main' method='GET'>" +  
"<button class='button' id='main'>Main</button>" +  
"</form>" +  
"</div>" +  
"<br>" +  
"</body>" +  
"</html>"  
 );  
 }   
}

// triviaapi   
// Account  
import jakarta.servlet.\*;  
import java.io.\*;  
public class Account extends HttpServlet {  
 public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 }  
  
 public void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {  
 String username = request.getParameter("user\_id");  
 String password = request.getParameter("password");  
 HttpSession session = request.getSession(true);  
 session.setAttribute("USER\_ID", username);   
 System.*out*.println("Logged in as:" + username);  
 response.setStatus(200);   
 }  
}  
  
// Trivia  
import jakarta.servlet.\*;  
import java.time.LocalDate;  
import java.io.\*;  
import java.lang.StringBuilder;  
import java.util.\*;  
@MultipartConfig  
public class Trivia extends HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response)  
 throws ServletException, IOException {  
 HttpSession session = request.getSession(false);  
 boolean isLoggedIn = isLoggedIn(request);  
 if (!isLoggedIn) {  
 response.setStatus(302);  
 response.sendRedirect("login");  
 } else {  
 response.setContentType("text/plain");  
 response.setCharacterEncoding("UTF-8");  
 //Use the path info and parameter map to find out the details of the get request  
 System.*out*.println(request.getPathInfo());  
 System.*out*.println(request.getParameterMap());  
 //use the resource Values to find out the details of the request and respond accordingly  
 File dir = new File("c:\\tomcat\\webapps\\triviaapi\\images");  
 String[] fileList = dir.list();  
 //use jason-io or Gson as opposed to buildng your own json array.  
 if (fileList != null && fileList.length > 0) {  
 String jsonArray = "[";  
 for (int i = 0; i < fileList.length; i++) {  
 jsonArray += fileList[i];  
 jsonArray += ",";  
 }  
 jsonArray += "]";  
 PrintWriter out = response.getWriter();   
 out.println(jsonArray);   
 }  
 response.setStatus(200);   
 }  
 }  
 protected void doPut(HttpServletRequest request, HttpServletResponse response)  
 throws ServletException, IOException {  
 //Use the path info and parameter map to find out the details of the PUT request  
 System.*out*.println(request.getPathInfo());  
 System.*out*.println(request.getParameterMap());  
 response.setStatus(200);  
 }  
 protected void doDelete(HttpServletRequest request, HttpServletResponse response)  
 throws ServletException, IOException {  
 //Use the path info and parameter map to find out the details of the DELETE request  
 System.*out*.println(request.getPathInfo());  
 System.*out*.println(request.getParameterMap());  
 response.setStatus(200);  
 }  
   
 @Override  
 protected void doPost(HttpServletRequest request, HttpServletResponse response)  
 throws ServletException, IOException {  
 System.*out*.println("In Do Post");  
 Part filePart = request.getPart("fileName");  
 String captionName = request.getParameter("caption");  
 String formDate = request.getParameter("date");  
 String fileName = filePart.getSubmittedFileName();  
  
 if(fileName.equals("")){  
 response.setStatus(302);  
 response.sendRedirect("upload");  
 return;  
 }  
 if(formDate.equals("")) formDate = "2020-10-10";  
 if(captionName.equals("")) captionName = "No caption";   
 filePart.write(System.*getProperty*("catalina.base") + "/webapps/triviaapi/images/" + fileName);  
 response.setStatus(200);  
 }   
 private boolean isLoggedIn(HttpServletRequest req) {  
 HttpSession session = req.getSession(false);  
 if (session == null || !req.isRequestedSessionIdValid()) {  
 return false;  
 }else{  
 return true;  
 }  
 }   
}  
  
// index.html  
<html>  
<body>  
<div id="login" style="display:block">  
 <label for="username">Username:</label><br>  
 <input type="text" id="username" name="username"/><br>  
 <label for="password">Password:</label><br>  
 <input type="text" id="password" name="password"/><br><br>  
 <button onclick="handleLogin()">login</button>  
</div>  
<div id="upload" style="display:none">  
 <p>Select a File:</p>  
<input type="file" id="myFileField" name="fileName" />  
 <button onclick="handleUpload()">Upload</buton>  
</div>  
<div id="index" style="display:block">  
 <p id="listView">List Files Here</p>  
 <button onclick="handleFileListing()">List</button>  
</div>  
</div>  
<div id="delete" style="display:block">  
 <button onclick="handleDelete()">Delete</button>  
</div>  
<script>  
function handleDelete() {  
 alert("Check the Tomcat Console Log to see if doDelete invoked");  
 const xhttp = new XMLHttpRequest();  
 xhttp.onreadystatechange = function() {  
 if (this.readyState == 4 && this.status == 200) {  
 *document*.getElementById("listView").innerHTML = this.responseText;  
 }  
 };  
 xhttp.open("Delete", "/triviaapi/trivia/Id=1");  
 xhttp.send();  
}  
function handleFileListing() {  
 const xhttp = new XMLHttpRequest();  
 xhttp.onreadystatechange = function() {  
 if (this.readyState == 4 && this.status == 200) {  
 *document*.getElementById("listView").innerHTML = this.responseText;  
 }  
 };  
 xhttp.open("GET", "/triviaapi/trivia/Id=1");  
 xhttp.send();  
}  
function handleLogin() {  
 const xhttp = new XMLHttpRequest();  
 xhttp.onload = function() {  
 *document*.getElementById("login").style.display = "none";  
 *document*.getElementById("upload").style.display = "block";  
 }  
 xhttp.open("POST", "/triviaapi/account?user\_id=temp&password=12345");  
 xhttp.send();  
}  
function handleUpload() {  
 var formData = new FormData();  
 formData.append("caption", "Test File");  
 formData.append("date", "2020-10-10");  
 formData.append("fileName", *document*.getElementById("myFileField").files[0]);  
 const xhr = new XMLHttpRequest();  
 xhr.onload = function() {  
 alert("File Uploaded");   
 }  
 xhr.open("POST", "/triviaapi/trivia");  
 xhr.send(formData);  
}  
</script>  
</body>  
</html>  
  
// AssignmentIaReference Code 2  
// ConsoleApp  
// Activity  
import java.io.\*;  
public class Activity {  
 public static void main(String[] args) throws IOException {  
 new Activity().onCreate();  
 }  
 public Activity() {  
 }  
 public void onCreate() {  
 AsyncTask UploadAsyncTask = new UploadAsyncTask().execute();   
 System.*out*.println("Waiting for Callback");  
 try {   
 BufferedReader br = new BufferedReader(new InputStreamReader(System.*in*));  
 br.readLine();  
 } catch (Exception e) { }  
 }  
}

// AsyncTask  
public abstract class AsyncTask {  
 AsyncTask execute() {   
 this.onPreExecute();  
 new WorkerThread(this).start();  
 return this;  
 }   
 protected abstract String doInBackground();  
 protected void onPreExecute() {  
 }  
 protected void onPostExecute(String result) {  
 }  
 protected void onProgressUpdate(String progress) {  
 }  
 protected void PublishProgress(String progress) {  
 }  
}

// UploadAsyncTask  
public class UploadAsyncTask extends AsyncTask {  
 protected void onPostExecute(String result) {  
 System.*out*.println(result);  
 }  
 protected String doInBackground() {  
 return new UploadClient().uploadFile();  
 }   
}

// UploadClient  
import java.io.\*;  
import java.net.\*;  
public class UploadClient {  
 public UploadClient() { }  
 public String uploadFile() {  
 String listing = "";  
 try {  
 Socket socket = new Socket("localhost", 8999);  
 BufferedReader in = new BufferedReader(  
 new InputStreamReader(socket.getInputStream()));  
 OutputStream out = socket.getOutputStream();  
 FileInputStream fis = new FileInputStream("AndroidLogo.png");  
 byte[] bytes = fis.readAllBytes();  
 out.write(bytes);  
 socket.shutdownOutput();  
 fis.close();  
 System.*out*.println("Came this far\n");  
 String filename = "";  
 while ((filename = in.readLine()) != null) {  
 listing += filename;  
 }  
 socket.shutdownInput();  
 } catch (Exception e) {  
 System.*err*.println(e);  
 }  
 return listing;  
 }  
}

// WorkerThread  
public class WorkerThread extends Thread {  
 private AsyncTask aTask = null;  
 public WorkerThread(AsyncTask asyncTask) {  
 this.aTask = asyncTask;  
 }  
 public void run() {  
 String result = aTask.doInBackground();  
 aTask.onPostExecute(result);  
   
 }  
}

// UploadServer  
// HttpServlet  
public abstract class HttpServlet {  
 protected void doGet(HttpServletRequest request, HttpServletResponse response) { return; };  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) { return; };  
}

// HttpServletRequest  
import java.io.\*;  
public class HttpServletRequest {  
 private InputStream inputStream = null;  
 public HttpServletRequest(InputStream inputStream) {  
 this.inputStream = inputStream;  
 }  
 public InputStream getInputStream() {return inputStream;}  
}

// HttpServletResponse  
import java.io.\*;  
public class HttpServletResponse {  
 private OutputStream outputStream = null;  
 public HttpServletResponse(OutputStream outputStream) {  
 this.outputStream = outputStream;  
 }  
 public OutputStream getOutputStream() {return outputStream;}  
}

// UploadServer  
import java.net.\*;  
import java.io.\*;  
public class UploadServer {  
 public static void main(String[] args) throws IOException {  
 ServerSocket serverSocket = null;  
 try {  
 serverSocket = new ServerSocket(8999);  
 } catch (IOException e) {  
 System.*err*.println("Could not listen on port: 8999.");  
 System.*exit*(-1);  
 }  
 while (true) {  
 new UploadServerThread(serverSocket.accept()).start();  
 }  
 }  
}

// UploadServerThread  
import java.net.\*;  
import java.io.\*;  
import java.time.Clock;  
public class UploadServerThread extends Thread {  
 private Socket socket = null;  
 public UploadServerThread(Socket socket) {  
 super("DirServerThread");  
 this.socket = socket;  
 }  
 public void run() {  
 try {  
 InputStream in = socket.getInputStream();   
 HttpServletRequest req = new HttpServletRequest(in);   
 OutputStream baos = new ByteArrayOutputStream();   
 HttpServletResponse res = new HttpServletResponse(baos);   
 HttpServlet httpServlet = new UploadServlet();  
 httpServlet.doPost(req, res);  
 OutputStream out = socket.getOutputStream();   
 out.write(((ByteArrayOutputStream) baos).toByteArray());  
 socket.close();  
 } catch (Exception e) { e.printStackTrace(); }  
 }  
}

// UploadServlet  
import java.io.\*;  
import java.time.Clock;  
public class UploadServlet extends HttpServlet {  
 protected void doPost(HttpServletRequest request, HttpServletResponse response) {  
 try {  
 InputStream in = request.getInputStream();   
 ByteArrayOutputStream baos = new ByteArrayOutputStream();   
 byte[] content = new byte[1];  
 int bytesRead = -1;   
 while( ( bytesRead = in.read( content ) ) != -1 ) {   
 baos.write( content, 0, bytesRead );   
 }  
 Clock clock = Clock.*systemDefaultZone*();  
 long milliSeconds=clock.millis();  
 OutputStream outputStream = new FileOutputStream(new File(String.*valueOf*(milliSeconds) + ".png"));  
 baos.writeTo(outputStream);  
 outputStream.close();  
 PrintWriter out = new PrintWriter(response.getOutputStream(), true);  
 File dir = new File(".");  
 String[] chld = dir.list();  
 for(int i = 0; i < chld.length; i++){  
 String fileName = chld[i];  
 out.println(fileName+"\n");  
 System.*out*.println(fileName);  
 }  
 } catch(Exception ex) {  
 System.*err*.println(ex);  
 }  
 }  
}

// DAO Pattern  
// DaoPattern  
import java.io.\*;  
import java.util.\*;  
  
// Create the base class with abstract serialize method that derived classes will need to implement  
abstract class Game {  
 String GameType;  
 public Game(String gameType) { this.GameType = gameType; }  
 abstract String serialize();  
}  
  
//Quiz class inherits from abstract Game class  
class Quiz extends Game {  
 int Id;  
 String Question;  
 String ContentPath;  
 public Quiz() { super("Quiz"); } //this is default constructor  
 public Quiz(String constructorParams) { // this constructor parses the constructorParams to set the object state  
 super("Quiz");  
 String[] keyvaluePairs = constructorParams.split(",");  
 for(int i = 0; i < keyvaluePairs.length; i++) {  
 String[] keyvaluePair = keyvaluePairs[i].split("=");  
 switch (keyvaluePair[0]) {  
 case "Id": this.Id = Integer.parseInt(keyvaluePair[1]); break;  
 case "Question": this.Question = keyvaluePair[1]; break;  
 case "ContentPath": this.ContentPath = keyvaluePair[1];break;  
 }  
 }   
 }  
 void setId(int id) { this.Id = id; }  
 void setQuestion (String question) { this.Question = question; }  
 void setContentPath (String contentPath) { this.ContentPath = contentPath; }  
 //Quiz's implementation of the serialize() method  
 String serialize() { return "Id="+this.Id+",Question="+this.Question+",ContentPath="+this.ContentPath; }  
}  
  
//Some other class that also inherits from abstract Game class.   
class Puzzle extends Game {  
 int Id;  
 String Name;  
 String Details;  
 public Puzzle() { super("Puzzle"); }  
 public Puzzle(String constructorParams) { super("Puzzle"); } //implementation not provided for now  
 void setId(int id) { this.Id = id; }  
 void setName(String name) { this.Name = name; }  
 void setDetails(String details) { this.Details = details; }  
 //Puzzle's implementation of the serialize method  
 String serialize() { return "Id="+this.Id+",Name="+this.Name+",Details="+this.Details; }  
}  
  
//An example of the Factory class  
// The GameFactory will be used by the Repository to create instances of different types of Game objects  
class GameFactory {  
 public Game createGame(String gameType, String constructorParameters) {  
 if (gameType.equalsIgnoreCase("Quiz")) {  
 return new Quiz(constructorParameters);  
 } else if (gameType.equalsIgnoreCase("Puzzle")) {  
 return new Puzzle(constructorParameters);  
 }  
 return null;  
 }  
}  
  
// IRepository interface  
// The interfaces allow for component based architecures and design   
interface IRepository   
{  
 public List<Game> select(String gameType, String criteria); //construct the Where clause using criteria deserialize each game object in   
 public void insert(Game game); //deserialize the game object and insert into database using JDBC  
}  
  
//The Repository class that implements IRepository.  
// Note that the Repository class can manage objects of any sub classes of Game class.  
// This is achieved using inheritance from Game class and the Factory design pattern  
class Repository implements IRepository {  
 List<Game> gameList = null;  
 public Repository() { gameList = new ArrayList<Game>(); }  
 @Override  
 public void insert(Game game) { gameList.add(game) ; }  
 @Override  
 public List<Game> select(String gameType, String pattern) { //note the use of Functional Programming in this method  
 Stream<Game> gameStream = gameList.stream();  
 Stream<Game> newGameStream = gameStream.filter(s -> s.GameType == gameType && s.serialize().contains(pattern));  
 List<Game> selectedGames = new ArrayList<Game>();  
 GameFactory gameFactory = new GameFactory();  
 newGameStream.forEach(s -> selectedGames.add(gameFactory.createGame(s.GameType, s.serialize())));  
 return selectedGames;  
 }  
}  
public class DaoPattern  
{  
 public static void main(String[] args) {  
 //repository object should be created in the constructor of the Servlet;  
 IRepository repository = new Repository();  
   
 //After extracting values from request object in the doPost or doPut method, create the quiz object and insert it into database via Repository  
 Quiz quiz = new Quiz(); quiz.setId(1);   
 quiz.setQuestion("Who is the Governal General of Canada"); quiz.setContentPath("youtube.com/xyz");  
 repository.insert(quiz);  
   
 //another doPost or doPut - another quiz object into the database  
 quiz = new Quiz(); quiz.setId(2);   
 quiz.setQuestion("Who is the Prime Minister of Canada"); quiz.setContentPath("youtube.com/abc");  
 repository.insert(quiz);  
  
 //Just to show that the above design enables Repository to manage any sub type of Game object  
 Puzzle puzzle = new Puzzle(); puzzle.setId(1); puzzle.setName("jigsaw");puzzle.setDetails("quite difficult");  
 repository.insert(puzzle);  
  
 //select some quiz(zes) from Repository in the doGet method if you like   
 List<Game> selectedGames = repository.select("Quiz", "Id=2");  
 for (Game game : selectedGames) {  
 System.out.println(game.serialize());  
 }  
 }  
}

// Game  
import java.io.\*;  
import java.util.\*;  
  
// Create the base class with abstract serialize method that derived classes will need to implement  
abstract class Game {  
 String GameType;  
 public Game(String gameType) { this.GameType = gameType; }  
 abstract String serialize();  
}  
  
//Quiz class inherits from abstract Game class  
class Quiz extends Game {  
 int Id;  
 String Question;  
 String ContentPath;  
 public Quiz() { super("Quiz"); } //this is default constructor  
 public Quiz(String constructorParams) { // this constructor parses the constructorParams to set the object state  
 super("Quiz");  
 String[] keyvaluePairs = constructorParams.split(",");  
 for(int i = 0; i < keyvaluePairs.length; i++) {  
 String[] keyvaluePair = keyvaluePairs[i].split("=");  
 switch (keyvaluePair[0]) {  
 case "Id": this.Id = Integer.*parseInt*(keyvaluePair[1]); break;  
 case "Question": this.Question = keyvaluePair[1]; break;  
 case "ContentPath": this.ContentPath = keyvaluePair[1];break;  
 }  
 }   
 }  
 void setId(int id) { this.Id = id; }  
 void setQuestion (String question) { this.Question = question; }  
 void setContentPath (String contentPath) { this.ContentPath = contentPath; }  
 //Quiz's implementation of the serialize() method  
 String serialize() { return "Id="+this.Id+",Question="+this.Question+",ContentPath="+this.ContentPath; }  
}  
  
//Some other class that also inherits from abstract Game class.   
class Puzzle extends Game {  
 int Id;  
 String Name;  
 String Details;  
 public Puzzle() { super("Puzzle"); }  
 public Puzzle(String constructorParams) { super("Puzzle"); } //implementation not provided for now  
 void setId(int id) { this.Id = id; }  
 void setName(String name) { this.Name = name; }  
 void setDetails(String details) { this.Details = details; }  
 //Puzzle's implementation of the serialize method  
 String serialize() { return "Id="+this.Id+",Name="+this.Name+",Details="+this.Details; }  
}  
  
//An example of the Factory class  
// The GameFactory will be used by the Repository to create instances of different types of Game objects  
class GameFactory {  
 public Game createGame(String gameType, String constructorParameters) {  
 if (gameType.equalsIgnoreCase("Quiz")) {  
 return new Quiz(constructorParameters);  
 } else if (gameType.equalsIgnoreCase("Puzzle")) {  
 return new Puzzle(constructorParameters);  
 }  
 return null;  
 }  
}  
  
// IRepository interface  
// The interfaces allow for component based architecures and design   
interface IRepository   
{  
 public List<Game> select(String gameType, String criteria); //construct the Where clause using criteria deserialize each game object in   
 public void insert(Game game); //deserialize the game object and insert into database using JDBC  
}  
  
//The Repository class that implements IRepository.  
// Note that the Repository class can manage objects of any sub classes of Game class.  
// This is achieved using inheritance from Game class and the Factory design pattern  
class Repository implements IRepository {  
 List<Game> gameList = null;  
 public Repository() { gameList = new ArrayList<Game>(); }  
 @Override  
 public void insert(Game game) { gameList.add(game) ; }  
 @Override  
 public List<Game> select(String gameType, String pattern) { //note the use of Functional Programming in this method  
 Stream<Game> gameStream = gameList.stream();  
 Stream<Game> newGameStream = gameStream.filter(s -> s.GameType == gameType && s.serialize().contains(pattern));  
 List<Game> selectedGames = new ArrayList<Game>();  
 GameFactory gameFactory = new GameFactory();  
 newGameStream.forEach(s -> selectedGames.add(gameFactory.createGame(s.GameType, s.serialize())));  
 return selectedGames;  
 }  
}  
public class DaoPattern  
{  
 public static void main(String[] args) {  
 //repository object should be created in the constructor of the Servlet;  
 IRepository repository = new Repository();  
   
 //After extracting values from request object in the doPost or doPut method, create the quiz object and insert it into database via Repository  
 Quiz quiz = new Quiz(); quiz.setId(1);   
 quiz.setQuestion("Who is the Governal General of Canada"); quiz.setContentPath("youtube.com/xyz");  
 repository.insert(quiz);  
   
 //another doPost or doPut - another quiz object into the database  
 quiz = new Quiz(); quiz.setId(2);   
 quiz.setQuestion("Who is the Prime Minister of Canada"); quiz.setContentPath("youtube.com/abc");  
 repository.insert(quiz);  
  
 //Just to show that the above design enables Repository to manage any sub type of Game object  
 Puzzle puzzle = new Puzzle(); puzzle.setId(1); puzzle.setName("jigsaw");puzzle.setDetails("quite difficult");  
 repository.insert(puzzle);  
  
 //select some quiz(zes) from Repository in the doGet method if you like   
 List<Game> selectedGames = repository.select("Quiz", "Id=2");  
 for (Game game : selectedGames) {  
 System.*out*.println(game.serialize());  
 }  
 }  
}

// GameFactory  
import java.io.\*;  
import java.util.\*;  
  
// Create the base class with abstract serialize method that derived classes will need to implement  
abstract class Game {  
 String GameType;  
 public Game(String gameType) { this.GameType = gameType; }  
 abstract String serialize();  
}  
  
//Quiz class inherits from abstract Game class  
class Quiz extends Game {  
 int Id;  
 String Question;  
 String ContentPath;  
 public Quiz() { super("Quiz"); } //this is default constructor  
 public Quiz(String constructorParams) { // this constructor parses the constructorParams to set the object state  
 super("Quiz");  
 String[] keyvaluePairs = constructorParams.split(",");  
 for(int i = 0; i < keyvaluePairs.length; i++) {  
 String[] keyvaluePair = keyvaluePairs[i].split("=");  
 switch (keyvaluePair[0]) {  
 case "Id": this.Id = Integer.*parseInt*(keyvaluePair[1]); break;  
 case "Question": this.Question = keyvaluePair[1]; break;  
 case "ContentPath": this.ContentPath = keyvaluePair[1];break;  
 }  
 }   
 }  
 void setId(int id) { this.Id = id; }  
 void setQuestion (String question) { this.Question = question; }  
 void setContentPath (String contentPath) { this.ContentPath = contentPath; }  
 //Quiz's implementation of the serialize() method  
 String serialize() { return "Id="+this.Id+",Question="+this.Question+",ContentPath="+this.ContentPath; }  
}  
  
//Some other class that also inherits from abstract Game class.   
class Puzzle extends Game {  
 int Id;  
 String Name;  
 String Details;  
 public Puzzle() { super("Puzzle"); }  
 public Puzzle(String constructorParams) { super("Puzzle"); } //implementation not provided for now  
 void setId(int id) { this.Id = id; }  
 void setName(String name) { this.Name = name; }  
 void setDetails(String details) { this.Details = details; }  
 //Puzzle's implementation of the serialize method  
 String serialize() { return "Id="+this.Id+",Name="+this.Name+",Details="+this.Details; }  
}  
  
//An example of the Factory class  
// The GameFactory will be used by the Repository to create instances of different types of Game objects  
class GameFactory {  
 public Game createGame(String gameType, String constructorParameters) {  
 if (gameType.equalsIgnoreCase("Quiz")) {  
 return new Quiz(constructorParameters);  
 } else if (gameType.equalsIgnoreCase("Puzzle")) {  
 return new Puzzle(constructorParameters);  
 }  
 return null;  
 }  
}  
  
// IRepository interface  
// The interfaces allow for component based architecures and design   
interface IRepository   
{  
 public List<Game> select(String gameType, String criteria); //construct the Where clause using criteria deserialize each game object in   
 public void insert(Game game); //deserialize the game object and insert into database using JDBC  
}  
  
//The Repository class that implements IRepository.  
// Note that the Repository class can manage objects of any sub classes of Game class.  
// This is achieved using inheritance from Game class and the Factory design pattern  
class Repository implements IRepository {  
 List<Game> gameList = null;  
 public Repository() { gameList = new ArrayList<Game>(); }  
 @Override  
 public void insert(Game game) { gameList.add(game) ; }  
 @Override  
 public List<Game> select(String gameType, String pattern) { //note the use of Functional Programming in this method  
 Stream<Game> gameStream = gameList.stream();  
 Stream<Game> newGameStream = gameStream.filter(s -> s.GameType == gameType && s.serialize().contains(pattern));  
 List<Game> selectedGames = new ArrayList<Game>();  
 GameFactory gameFactory = new GameFactory();  
 newGameStream.forEach(s -> selectedGames.add(gameFactory.createGame(s.GameType, s.serialize())));  
 return selectedGames;  
 }  
}  
public class DaoPattern  
{  
 public static void main(String[] args) {  
 //repository object should be created in the constructor of the Servlet;  
 IRepository repository = new Repository();  
   
 //After extracting values from request object in the doPost or doPut method, create the quiz object and insert it into database via Repository  
 Quiz quiz = new Quiz(); quiz.setId(1);   
 quiz.setQuestion("Who is the Governal General of Canada"); quiz.setContentPath("youtube.com/xyz");  
 repository.insert(quiz);  
   
 //another doPost or doPut - another quiz object into the database  
 quiz = new Quiz(); quiz.setId(2);   
 quiz.setQuestion("Who is the Prime Minister of Canada"); quiz.setContentPath("youtube.com/abc");  
 repository.insert(quiz);  
  
 //Just to show that the above design enables Repository to manage any sub type of Game object  
 Puzzle puzzle = new Puzzle(); puzzle.setId(1); puzzle.setName("jigsaw");puzzle.setDetails("quite difficult");  
 repository.insert(puzzle);  
  
 //select some quiz(zes) from Repository in the doGet method if you like   
 List<Game> selectedGames = repository.select("Quiz", "Id=2");  
 for (Game game : selectedGames) {  
 System.*out*.println(game.serialize());  
 }  
 }  
}

// IRepository  
import java.io.\*;  
import java.util.\*;  
  
// Create the base class with abstract serialize method that derived classes will need to implement  
abstract class Game {  
 String GameType;  
 public Game(String gameType) { this.GameType = gameType; }  
 abstract String serialize();  
}  
  
//Quiz class inherits from abstract Game class  
class Quiz extends Game {  
 int Id;  
 String Question;  
 String ContentPath;  
 public Quiz() { super("Quiz"); } //this is default constructor  
 public Quiz(String constructorParams) { // this constructor parses the constructorParams to set the object state  
 super("Quiz");  
 String[] keyvaluePairs = constructorParams.split(",");  
 for(int i = 0; i < keyvaluePairs.length; i++) {  
 String[] keyvaluePair = keyvaluePairs[i].split("=");  
 switch (keyvaluePair[0]) {  
 case "Id": this.Id = Integer.*parseInt*(keyvaluePair[1]); break;  
 case "Question": this.Question = keyvaluePair[1]; break;  
 case "ContentPath": this.ContentPath = keyvaluePair[1];break;  
 }  
 }   
 }  
 void setId(int id) { this.Id = id; }  
 void setQuestion (String question) { this.Question = question; }  
 void setContentPath (String contentPath) { this.ContentPath = contentPath; }  
 //Quiz's implementation of the serialize() method  
 String serialize() { return "Id="+this.Id+",Question="+this.Question+",ContentPath="+this.ContentPath; }  
}  
  
//Some other class that also inherits from abstract Game class.   
class Puzzle extends Game {  
 int Id;  
 String Name;  
 String Details;  
 public Puzzle() { super("Puzzle"); }  
 public Puzzle(String constructorParams) { super("Puzzle"); } //implementation not provided for now  
 void setId(int id) { this.Id = id; }  
 void setName(String name) { this.Name = name; }  
 void setDetails(String details) { this.Details = details; }  
 //Puzzle's implementation of the serialize method  
 String serialize() { return "Id="+this.Id+",Name="+this.Name+",Details="+this.Details; }  
}  
  
//An example of the Factory class  
// The GameFactory will be used by the Repository to create instances of different types of Game objects  
class GameFactory {  
 public Game createGame(String gameType, String constructorParameters) {  
 if (gameType.equalsIgnoreCase("Quiz")) {  
 return new Quiz(constructorParameters);  
 } else if (gameType.equalsIgnoreCase("Puzzle")) {  
 return new Puzzle(constructorParameters);  
 }  
 return null;  
 }  
}  
  
// IRepository interface  
// The interfaces allow for component based architecures and design   
interface IRepository   
{  
 public List<Game> select(String gameType, String criteria); //construct the Where clause using criteria deserialize each game object in   
 public void insert(Game game); //deserialize the game object and insert into database using JDBC  
}  
  
//The Repository class that implements IRepository.  
// Note that the Repository class can manage objects of any sub classes of Game class.  
// This is achieved using inheritance from Game class and the Factory design pattern  
class Repository implements IRepository {  
 List<Game> gameList = null;  
 public Repository() { gameList = new ArrayList<Game>(); }  
 @Override  
 public void insert(Game game) { gameList.add(game) ; }  
 @Override  
 public List<Game> select(String gameType, String pattern) { //note the use of Functional Programming in this method  
 Stream<Game> gameStream = gameList.stream();  
 Stream<Game> newGameStream = gameStream.filter(s -> s.GameType == gameType && s.serialize().contains(pattern));  
 List<Game> selectedGames = new ArrayList<Game>();  
 GameFactory gameFactory = new GameFactory();  
 newGameStream.forEach(s -> selectedGames.add(gameFactory.createGame(s.GameType, s.serialize())));  
 return selectedGames;  
 }  
}  
public class DaoPattern  
{  
 public static void main(String[] args) {  
 //repository object should be created in the constructor of the Servlet;  
 IRepository repository = new Repository();  
   
 //After extracting values from request object in the doPost or doPut method, create the quiz object and insert it into database via Repository  
 Quiz quiz = new Quiz(); quiz.setId(1);   
 quiz.setQuestion("Who is the Governal General of Canada"); quiz.setContentPath("youtube.com/xyz");  
 repository.insert(quiz);  
   
 //another doPost or doPut - another quiz object into the database  
 quiz = new Quiz(); quiz.setId(2);   
 quiz.setQuestion("Who is the Prime Minister of Canada"); quiz.setContentPath("youtube.com/abc");  
 repository.insert(quiz);  
  
 //Just to show that the above design enables Repository to manage any sub type of Game object  
 Puzzle puzzle = new Puzzle(); puzzle.setId(1); puzzle.setName("jigsaw");puzzle.setDetails("quite difficult");  
 repository.insert(puzzle);  
  
 //select some quiz(zes) from Repository in the doGet method if you like   
 List<Game> selectedGames = repository.select("Quiz", "Id=2");  
 for (Game game : selectedGames) {  
 System.*out*.println(game.serialize());  
 }  
 }  
}

// Puzzle  
import java.io.\*;  
import java.util.\*;  
  
// Create the base class with abstract serialize method that derived classes will need to implement  
abstract class Game {  
 String GameType;  
 public Game(String gameType) { this.GameType = gameType; }  
 abstract String serialize();  
}  
  
//Quiz class inherits from abstract Game class  
class Quiz extends Game {  
 int Id;  
 String Question;  
 String ContentPath;  
 public Quiz() { super("Quiz"); } //this is default constructor  
 public Quiz(String constructorParams) { // this constructor parses the constructorParams to set the object state  
 super("Quiz");  
 String[] keyvaluePairs = constructorParams.split(",");  
 for(int i = 0; i < keyvaluePairs.length; i++) {  
 String[] keyvaluePair = keyvaluePairs[i].split("=");  
 switch (keyvaluePair[0]) {  
 case "Id": this.Id = Integer.*parseInt*(keyvaluePair[1]); break;  
 case "Question": this.Question = keyvaluePair[1]; break;  
 case "ContentPath": this.ContentPath = keyvaluePair[1];break;  
 }  
 }   
 }  
 void setId(int id) { this.Id = id; }  
 void setQuestion (String question) { this.Question = question; }  
 void setContentPath (String contentPath) { this.ContentPath = contentPath; }  
 //Quiz's implementation of the serialize() method  
 String serialize() { return "Id="+this.Id+",Question="+this.Question+",ContentPath="+this.ContentPath; }  
}  
  
//Some other class that also inherits from abstract Game class.   
class Puzzle extends Game {  
 int Id;  
 String Name;  
 String Details;  
 public Puzzle() { super("Puzzle"); }  
 public Puzzle(String constructorParams) { super("Puzzle"); } //implementation not provided for now  
 void setId(int id) { this.Id = id; }  
 void setName(String name) { this.Name = name; }  
 void setDetails(String details) { this.Details = details; }  
 //Puzzle's implementation of the serialize method  
 String serialize() { return "Id="+this.Id+",Name="+this.Name+",Details="+this.Details; }  
}  
  
//An example of the Factory class  
// The GameFactory will be used by the Repository to create instances of different types of Game objects  
class GameFactory {  
 public Game createGame(String gameType, String constructorParameters) {  
 if (gameType.equalsIgnoreCase("Quiz")) {  
 return new Quiz(constructorParameters);  
 } else if (gameType.equalsIgnoreCase("Puzzle")) {  
 return new Puzzle(constructorParameters);  
 }  
 return null;  
 }  
}  
  
// IRepository interface  
// The interfaces allow for component based architecures and design   
interface IRepository   
{  
 public List<Game> select(String gameType, String criteria); //construct the Where clause using criteria deserialize each game object in   
 public void insert(Game game); //deserialize the game object and insert into database using JDBC  
}  
  
//The Repository class that implements IRepository.  
// Note that the Repository class can manage objects of any sub classes of Game class.  
// This is achieved using inheritance from Game class and the Factory design pattern  
class Repository implements IRepository {  
 List<Game> gameList = null;  
 public Repository() { gameList = new ArrayList<Game>(); }  
 @Override  
 public void insert(Game game) { gameList.add(game) ; }  
 @Override  
 public List<Game> select(String gameType, String pattern) { //note the use of Functional Programming in this method  
 Stream<Game> gameStream = gameList.stream();  
 Stream<Game> newGameStream = gameStream.filter(s -> s.GameType == gameType && s.serialize().contains(pattern));  
 List<Game> selectedGames = new ArrayList<Game>();  
 GameFactory gameFactory = new GameFactory();  
 newGameStream.forEach(s -> selectedGames.add(gameFactory.createGame(s.GameType, s.serialize())));  
 return selectedGames;  
 }  
}  
public class DaoPattern  
{  
 public static void main(String[] args) {  
 //repository object should be created in the constructor of the Servlet;  
 IRepository repository = new Repository();  
   
 //After extracting values from request object in the doPost or doPut method, create the quiz object and insert it into database via Repository  
 Quiz quiz = new Quiz(); quiz.setId(1);   
 quiz.setQuestion("Who is the Governal General of Canada"); quiz.setContentPath("youtube.com/xyz");  
 repository.insert(quiz);  
   
 //another doPost or doPut - another quiz object into the database  
 quiz = new Quiz(); quiz.setId(2);   
 quiz.setQuestion("Who is the Prime Minister of Canada"); quiz.setContentPath("youtube.com/abc");  
 repository.insert(quiz);  
  
 //Just to show that the above design enables Repository to manage any sub type of Game object  
 Puzzle puzzle = new Puzzle(); puzzle.setId(1); puzzle.setName("jigsaw");puzzle.setDetails("quite difficult");  
 repository.insert(puzzle);  
  
 //select some quiz(zes) from Repository in the doGet method if you like   
 List<Game> selectedGames = repository.select("Quiz", "Id=2");  
 for (Game game : selectedGames) {  
 System.*out*.println(game.serialize());  
 }  
 }  
}

// Quiz  
import java.io.\*;  
import java.util.\*;  
  
// Create the base class with abstract serialize method that derived classes will need to implement  
abstract class Game {  
 String GameType;  
 public Game(String gameType) { this.GameType = gameType; }  
 abstract String serialize();  
}  
  
//Quiz class inherits from abstract Game class  
class Quiz extends Game {  
 int Id;  
 String Question;  
 String ContentPath;  
 public Quiz() { super("Quiz"); } //this is default constructor  
 public Quiz(String constructorParams) { // this constructor parses the constructorParams to set the object state  
 super("Quiz");  
 String[] keyvaluePairs = constructorParams.split(",");  
 for(int i = 0; i < keyvaluePairs.length; i++) {  
 String[] keyvaluePair = keyvaluePairs[i].split("=");  
 switch (keyvaluePair[0]) {  
 case "Id": this.Id = Integer.*parseInt*(keyvaluePair[1]); break;  
 case "Question": this.Question = keyvaluePair[1]; break;  
 case "ContentPath": this.ContentPath = keyvaluePair[1];break;  
 }  
 }   
 }  
 void setId(int id) { this.Id = id; }  
 void setQuestion (String question) { this.Question = question; }  
 void setContentPath (String contentPath) { this.ContentPath = contentPath; }  
 //Quiz's implementation of the serialize() method  
 String serialize() { return "Id="+this.Id+",Question="+this.Question+",ContentPath="+this.ContentPath; }  
}  
  
//Some other class that also inherits from abstract Game class.   
class Puzzle extends Game {  
 int Id;  
 String Name;  
 String Details;  
 public Puzzle() { super("Puzzle"); }  
 public Puzzle(String constructorParams) { super("Puzzle"); } //implementation not provided for now  
 void setId(int id) { this.Id = id; }  
 void setName(String name) { this.Name = name; }  
 void setDetails(String details) { this.Details = details; }  
 //Puzzle's implementation of the serialize method  
 String serialize() { return "Id="+this.Id+",Name="+this.Name+",Details="+this.Details; }  
}  
  
//An example of the Factory class  
// The GameFactory will be used by the Repository to create instances of different types of Game objects  
class GameFactory {  
 public Game createGame(String gameType, String constructorParameters) {  
 if (gameType.equalsIgnoreCase("Quiz")) {  
 return new Quiz(constructorParameters);  
 } else if (gameType.equalsIgnoreCase("Puzzle")) {  
 return new Puzzle(constructorParameters);  
 }  
 return null;  
 }  
}  
  
// IRepository interface  
// The interfaces allow for component based architecures and design   
interface IRepository   
{  
 public List<Game> select(String gameType, String criteria); //construct the Where clause using criteria deserialize each game object in   
 public void insert(Game game); //deserialize the game object and insert into database using JDBC  
}  
  
//The Repository class that implements IRepository.  
// Note that the Repository class can manage objects of any sub classes of Game class.  
// This is achieved using inheritance from Game class and the Factory design pattern  
class Repository implements IRepository {  
 List<Game> gameList = null;  
 public Repository() { gameList = new ArrayList<Game>(); }  
 @Override  
 public void insert(Game game) { gameList.add(game) ; }  
 @Override  
 public List<Game> select(String gameType, String pattern) { //note the use of Functional Programming in this method  
 Stream<Game> gameStream = gameList.stream();  
 Stream<Game> newGameStream = gameStream.filter(s -> s.GameType == gameType && s.serialize().contains(pattern));  
 List<Game> selectedGames = new ArrayList<Game>();  
 GameFactory gameFactory = new GameFactory();  
 newGameStream.forEach(s -> selectedGames.add(gameFactory.createGame(s.GameType, s.serialize())));  
 return selectedGames;  
 }  
}  
public class DaoPattern  
{  
 public static void main(String[] args) {  
 //repository object should be created in the constructor of the Servlet;  
 IRepository repository = new Repository();  
   
 //After extracting values from request object in the doPost or doPut method, create the quiz object and insert it into database via Repository  
 Quiz quiz = new Quiz(); quiz.setId(1);   
 quiz.setQuestion("Who is the Governal General of Canada"); quiz.setContentPath("youtube.com/xyz");  
 repository.insert(quiz);  
   
 //another doPost or doPut - another quiz object into the database  
 quiz = new Quiz(); quiz.setId(2);   
 quiz.setQuestion("Who is the Prime Minister of Canada"); quiz.setContentPath("youtube.com/abc");  
 repository.insert(quiz);  
  
 //Just to show that the above design enables Repository to manage any sub type of Game object  
 Puzzle puzzle = new Puzzle(); puzzle.setId(1); puzzle.setName("jigsaw");puzzle.setDetails("quite difficult");  
 repository.insert(puzzle);  
  
 //select some quiz(zes) from Repository in the doGet method if you like   
 List<Game> selectedGames = repository.select("Quiz", "Id=2");  
 for (Game game : selectedGames) {  
 System.*out*.println(game.serialize());  
 }  
 }  
}

// Repository  
import java.io.\*;  
import java.util.\*;  
  
// Create the base class with abstract serialize method that derived classes will need to implement  
abstract class Game {  
 String GameType;  
 public Game(String gameType) { this.GameType = gameType; }  
 abstract String serialize();  
}  
  
//Quiz class inherits from abstract Game class  
class Quiz extends Game {  
 int Id;  
 String Question;  
 String ContentPath;  
 public Quiz() { super("Quiz"); } //this is default constructor  
 public Quiz(String constructorParams) { // this constructor parses the constructorParams to set the object state  
 super("Quiz");  
 String[] keyvaluePairs = constructorParams.split(",");  
 for(int i = 0; i < keyvaluePairs.length; i++) {  
 String[] keyvaluePair = keyvaluePairs[i].split("=");  
 switch (keyvaluePair[0]) {  
 case "Id": this.Id = Integer.*parseInt*(keyvaluePair[1]); break;  
 case "Question": this.Question = keyvaluePair[1]; break;  
 case "ContentPath": this.ContentPath = keyvaluePair[1];break;  
 }  
 }   
 }  
 void setId(int id) { this.Id = id; }  
 void setQuestion (String question) { this.Question = question; }  
 void setContentPath (String contentPath) { this.ContentPath = contentPath; }  
 //Quiz's implementation of the serialize() method  
 String serialize() { return "Id="+this.Id+",Question="+this.Question+",ContentPath="+this.ContentPath; }  
}  
  
//Some other class that also inherits from abstract Game class.   
class Puzzle extends Game {  
 int Id;  
 String Name;  
 String Details;  
 public Puzzle() { super("Puzzle"); }  
 public Puzzle(String constructorParams) { super("Puzzle"); } //implementation not provided for now  
 void setId(int id) { this.Id = id; }  
 void setName(String name) { this.Name = name; }  
 void setDetails(String details) { this.Details = details; }  
 //Puzzle's implementation of the serialize method  
 String serialize() { return "Id="+this.Id+",Name="+this.Name+",Details="+this.Details; }  
}  
  
//An example of the Factory class  
// The GameFactory will be used by the Repository to create instances of different types of Game objects  
class GameFactory {  
 public Game createGame(String gameType, String constructorParameters) {  
 if (gameType.equalsIgnoreCase("Quiz")) {  
 return new Quiz(constructorParameters);  
 } else if (gameType.equalsIgnoreCase("Puzzle")) {  
 return new Puzzle(constructorParameters);  
 }  
 return null;  
 }  
}  
  
// IRepository interface  
// The interfaces allow for component based architecures and design   
interface IRepository   
{  
 public List<Game> select(String gameType, String criteria); //construct the Where clause using criteria deserialize each game object in   
 public void insert(Game game); //deserialize the game object and insert into database using JDBC  
}  
  
//The Repository class that implements IRepository.  
// Note that the Repository class can manage objects of any sub classes of Game class.  
// This is achieved using inheritance from Game class and the Factory design pattern  
class Repository implements IRepository {  
 List<Game> gameList = null;  
 public Repository() { gameList = new ArrayList<Game>(); }  
 @Override  
 public void insert(Game game) { gameList.add(game) ; }  
 @Override  
 public List<Game> select(String gameType, String pattern) { //note the use of Functional Programming in this method  
 Stream<Game> gameStream = gameList.stream();  
 Stream<Game> newGameStream = gameStream.filter(s -> s.GameType == gameType && s.serialize().contains(pattern));  
 List<Game> selectedGames = new ArrayList<Game>();  
 GameFactory gameFactory = new GameFactory();  
 newGameStream.forEach(s -> selectedGames.add(gameFactory.createGame(s.GameType, s.serialize())));  
 return selectedGames;  
 }  
}  
public class DaoPattern  
{  
 public static void main(String[] args) {  
 //repository object should be created in the constructor of the Servlet;  
 IRepository repository = new Repository();  
   
 //After extracting values from request object in the doPost or doPut method, create the quiz object and insert it into database via Repository  
 Quiz quiz = new Quiz(); quiz.setId(1);   
 quiz.setQuestion("Who is the Governal General of Canada"); quiz.setContentPath("youtube.com/xyz");  
 repository.insert(quiz);  
   
 //another doPost or doPut - another quiz object into the database  
 quiz = new Quiz(); quiz.setId(2);   
 quiz.setQuestion("Who is the Prime Minister of Canada"); quiz.setContentPath("youtube.com/abc");  
 repository.insert(quiz);  
  
 //Just to show that the above design enables Repository to manage any sub type of Game object  
 Puzzle puzzle = new Puzzle(); puzzle.setId(1); puzzle.setName("jigsaw");puzzle.setDetails("quite difficult");  
 repository.insert(puzzle);  
  
 //select some quiz(zes) from Repository in the doGet method if you like   
 List<Game> selectedGames = repository.select("Quiz", "Id=2");  
 for (Game game : selectedGames) {  
 System.*out*.println(game.serialize());  
 }  
 }  
}

// ThreadSafety  
public class ThreadSafety implements Runnable{  
 int shared = 2;  
 public static void main(String[] args) {  
 ThreadSafety ts = new ThreadSafety();  
 Thread t1 = new Thread(ts, "T1");  
 Thread t2 = new Thread(ts, "T2");  
 t1.start();  
 t2.start();  
 }  
 public void run() {  
 int copy = shared;  
 try {   
 Thread.*sleep*((int)(Math.*random*() \* 5000));   
 } catch (InterruptedException e) { }  
 shared = copy - 1;  
 System.*out*.println(Thread.*currentThread*().getName() + ": " +shared);  
 }   
}

Two threads (T1 and T2) manipulate a shared variable shared without synchronization, leading to race conditions. Each thread copies the current value of shared, sleeps for a random time, and then decrements it. Due to the unpredictable timing of thread execution and the lack of thread safety mechanisms, the threads may read and write shared at different times, resulting in various possible outputs such as both threads printing 1, one printing 1 and the other 0, or both printing 0.

Synchronization (public synchronized void run()) fixes the issue by ensuring that only one thread can access and modify the shared variable shared at a time, thus preventing race conditions. When the code that reads and writes to shared is enclosed within a synchronized block or method, it creates a mutual exclusion (mutex) lock. This means if one thread is executing the synchronized code, other threads must wait until the lock is released before they can enter that block. As a result, each thread reads the most up-to-date value of shared, performs its computation without interference, and writes back the result safely. This coordinated access eliminates the unpredictability caused by concurrent modifications, ensuring consistent and expected outputs from the threads.

// SimpleThread  
public class SimpleThread extends Thread {  
 public SimpleThread(String str) {  
 super(str);  
 }  
  
 public void run() {  
 for (int i = 0; i < 10; i++) {  
 System.*out*.println(i + " " + getName());  
 try {  
 *sleep*((long)(Math.*random*() \* 1000));  
 } catch (InterruptedException e) {}  
 }  
 System.*out*.println("DONE! " + getName());  
 }  
}

// TwoThreadsTest  
public class TwoThreadsTest {  
 public static void main (String[] args) {  
 new SimpleThread("Jamaica").start();  
 new SimpleThread("Fiji").start();  
 }  
}

// SemaphoreTest  
import java.util.concurrent.Semaphore;  
public class SemaphoreTest {  
 // max 4 people  
 static Semaphore *semaphore* = new Semaphore(4);  
// static Semaphore sem = new Semaphore(3);  
 static class MyATMThread extends Thread {  
 String name = "";  
 MyATMThread(String name) {  
 this.name = name;  
 }  
 public void run() {  
 try {  
 System.*out*.println(name + " : acquiring lock...");  
 System.*out*.println(name + " : available Semaphore permits now: "  
 + *semaphore*.availablePermits());  
 *semaphore*.acquire();  
 System.*out*.println(name + " : got the permit!");  
 try {  
 for (int i = 1; i <= 5; i++) {  
 System.*out*.println(name + " : is performing operation " + i  
 + ", available Semaphore permits : "  
 + *semaphore*.availablePermits());  
 // sleep 1 second  
 Thread.*sleep*(1000);  
 }  
 } finally {  
 // calling release() after a successful acquire()  
 System.*out*.println(name + " : releasing lock...");  
 *semaphore*.release();  
 System.*out*.println(name + " : available Semaphore permits now: "  
 + *semaphore*.availablePermits());  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 System.*out*.println("Total available Semaphore permits : "  
 + *semaphore*.availablePermits());  
 MyATMThread t1 = new MyATMThread("A");  
 t1.start();  
 MyATMThread t2 = new MyATMThread("B");  
 t2.start();  
 MyATMThread t3 = new MyATMThread("C");  
 t3.start();  
 MyATMThread t4 = new MyATMThread("D");  
 t4.start();  
 MyATMThread t5 = new MyATMThread("E");  
 t5.start();  
 MyATMThread t6 = new MyATMThread("F");  
 t6.start();  
 }  
}

// Reminder  
import java.util.Timer;  
import java.util.TimerTask;  
*/\*\*  
\* Simple demo that uses java.util.Timer to schedule a task  
\* to execute once 5 seconds have passed.  
\*/*public class Reminder {  
Timer timer;  
public Reminder(int seconds) {  
timer = new Timer();  
timer.schedule(new RemindTask(), seconds\*1000);  
}  
class RemindTask extends TimerTask {  
public void run() {  
System.*out*.println("Time's up!");  
timer.cancel(); //Terminate the timer thread  
}  
}  
public static void main(String args[]) {  
new Reminder(5);  
System.*out*.println("Task scheduled.");  
}  
}

// ProducerConsumerTest  
public class ProducerConsumerTest {  
 public static void main(String[] args) {  
 CubbyHole c1 = new CubbyHole();  
 Producer p11 = new Producer(c1, 1);  
 Consumer c11 = new Consumer(c1, 1);  
 CubbyHole c2 = new CubbyHole();  
 Producer p12 = new Producer(c2, 2);  
 Consumer c12 = new Consumer(c2, 2);  
 p11.start();  
 c11.start();  
 p12.start();  
 c12.start();  
 }  
}

// Producer  
public class Producer extends Thread {  
 private CubbyHole cubbyhole;  
 private int number;  
 public Producer(CubbyHole c, int number) {  
 cubbyhole = c;  
 this.number = number;  
 }  
 public void run() {  
 for (int i = 0; i < 10; i++) {  
 cubbyhole.put(number, i);  
 try {  
 *sleep*((int)(Math.*random*() \* 100));  
 } catch (InterruptedException e) { }  
 }  
 }  
}

// CubbyHole  
public class CubbyHole {  
 private int contents;  
 private boolean available = false;  
   
 public synchronized int get(int who) {  
 while (available == false) {  
 try {  
 wait();  
 } catch (InterruptedException e) { }  
 }  
 available = false;  
 System.*out*.println("Consumer " + who + " got: " + contents);  
 notifyAll();  
 return contents;  
 }  
  
 public synchronized void put(int who, int value) {  
 while (available == true) {  
 try {  
 wait();  
 } catch (InterruptedException e) { }  
 }  
 contents = value;  
 available = true;  
 System.*out*.println("Producer " + who + " put: " + contents);  
 notifyAll();  
 }  
}

// Consumer  
public class Consumer extends Thread {  
 private CubbyHole cubbyhole;  
 private int number;  
 public Consumer(CubbyHole c, int number) {  
 cubbyhole = c;  
 this.number = number;  
 }  
 public void run() {  
 int value = 0;  
 for (int i = 0; i < 10; i++) {  
 value = cubbyhole.get(number);  
 }  
 }  
}

// AnnoyingBeep  
import java.util.Timer;  
import java.util.TimerTask;  
import java.awt.Toolkit;  
*/\*\*  
\* Schedule a task that executes once every second.  
\*/*public class AnnoyingBeep {  
 Toolkit toolkit;  
 Timer timer;  
 public AnnoyingBeep() {  
 toolkit = Toolkit.*getDefaultToolkit*();  
 timer = new Timer();  
 timer.schedule(new RemindTask(), 0, 1\*1000); //subsequent rate  
 }  
 class RemindTask extends TimerTask {  
 int numWarningBeeps = 3;  
 public void run() {  
 if (numWarningBeeps > 0) {  
 toolkit.beep();  
 System.*out*.println("Beep!");  
 numWarningBeeps--;  
 } else {  
 toolkit.beep();  
 System.*out*.println("Time's up!");  
 //timer.cancel(); //Not necessary because we call  
 System.*exit*(0);  
 }  
 }  
 }  
 public static void main(String args[]) {  
 System.*out*.println("About to schedule task.");  
 new AnnoyingBeep();  
 System.*out*.println("Task scheduled.");  
 }  
}

//1) Hash the Password  
<!DOCTYPE html>  
<html>  
<head>  
<script src="https://cdn.jsdelivr.net/npm/md5-js-tools@1.0.2/lib/md5.min.js"></script>  
</head>  
<body>  
<h2>HTML Forms</h2>  
<form action="/action\_page.php" >  
 <label for="fname">First name:</label><br>  
<input type="text" id="fname" name="fname" value="John"><br>  
 <label for="lname">Last name:</label><br>  
<input type="text" id="lname" name="lname" value="Doe"><br><br>  
<input type="submit" value="Submit" id="myBtn" >  
</form>   
<script> const element = document.getElementById("myBtn");  
element.addEventListener("click", handleLogin);  
function handleLogin() {  
var password = document.getElementById("lname").value;  
var hash = MD5.generate(password);  
document.getElementById("lname").value = hash;}  
</script>  
<p>If you click the "Submit" button, the form-data will be sent to a page called "/action\_page.php".</p>  
</body></html>  
// 2)Is it better to hash the password on the client side or on the server side? How would your answer be influenced by whether your Tomcat Server is SSL enabled or not  
If SSL is enabled the hashing could be done on the client or server side as the password will be encrypted. Otherwise hashing on the client side can at least prevent intruder stealing the password.  
// 3) Modify the LoginServlet further such that the Form element of the login page now uses a regular HTML Button (i.e. <button type=\"button\") but the login page is still posted to the server.  
<input type="button" value="SignIn" onclick="handleLogin()" /></form>  
<script>  
function handleLogin() {  
var password = document.getElementById("username").value;  
// preprocess password here  
document.getElementById("loginForm").submit();}  
</script>  
// 4) Rewrite the MainServlet so that it generates a landing page that is now composed of only one Form element but this Form element now contains four submit buttons labelled UPLOAD, GALLERY, SEARCH and LOGOUT. Pressing any of these submit buttons shall now facilitate navigation to the respective page. Copy and paste the revised MainServlet (and other impacted code, if any) below.  
<form action="main" method="POST">  
<input type='submit' name="nav" value='upload' />  
<input type='submit' name="nav" value='search' />  
<input type='submit' name="nav" value='gallery' />  
<input type='submit' name="nav" value='logout' />  
</form>   
//in servlet   
String nav = request.getParameter("nav");  
if (nav == null) {  
//no button has been selected  
} else if (nav.equals("upload")) {  
//upload button was pressed  
Response.sendRedirect(“upload”);  
} else if (nav.equals("gallery")) {  
//gallery button was pressed  
} else if (nav.equals("search")){//search button was pressed  
} else if (nav.equals("logout")) {  
//logout button was pressed  
} else {//someone has altered the HTML and sent a different value}  
//5)The onCreate() method of the Activity class in ConsoleApp.zip calls the uploadFile() method of the UploadClient, and thereafter prints out the received directory listing, synchronously. Define an UploadAsyncTask class, that inherits from AsyncTask, and implements its appropriate methods. The revised Activity class listed below shows the use of this class in its onCreate() in calling the uploadFile() method asynchronously. In other words, most of the times, when you run the revised client you should see the line “Waiting for Callback” before seeing the directory listing on the console. Please note that your console app is now composed of this revised Activity class listed below, the UploadAsyncTask that you are asked to create, AsyncTask and WorkerThread classes of the AsyncTask.zip, and UploadClient class from before. Copy and paste your UploadAsyncTask class below.  
import java.io.\*;  
public class Activity {  
public static void main(String[] args) throws IOException {  
new Activity().onCreate();  
}  
public Activity() {  
}  
public void onCreate() {  
AsyncTask UploadAsyncTask = new UploadAsyncTask().execute(); System.out.println("Waiting for Callback");  
try {   
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
br.readLine();} catch (Exception e) { }  
}}   
//5) Solution  
public class UploadAsyncTask extends AsyncTask {  
protected void onPostExecute(String result) {  
System.out.println(result);  
}  
protected String doInBackground() {  
return new UploadClient().uploadFile();}}  
//6) Refer to the ThreadSafety class in the ThreadSafety.zip attached to the Midterm Guidelines folder to answer the following questions.   
// ThreadSafety  
public class ThreadSafety implements Runnable{  
int shared = 0;  
public static void main(String[] args) {  
ThreadSafety ts = new ThreadSafety();  
Thread t1 = new Thread(ts, "T1");  
Thread t2 = new Thread(ts, "T2");  
t1.start();  
t2.start();  
}  
public void run() {  
int copy = shared;  
try {   
Thread.sleep((int)(Math.random() \* 5000));   
} catch (InterruptedException e) { }  
shared = copy + 1;  
System.out.println(Thread.currentThread().getName() + ": " +shared);}}  
Q6-a) What are all possible outputs of the following java code?  
T1 :1, T2: 2  
T2: 1, T1: 2  
T1: 1, T2:1  
T2:1, T1: 1  
T1:2 and T2:2 is also possible due to the timing of system.out.println calls in the two threads  
Q6-b) What are the possible output(s) of a revised ThreadSafety class listed below.  
 public void run() {  
synchronized(this) {  
int copy = shared;  
try {   
Thread.sleep((int)(Math.random() \* 5000));   
} catch (InterruptedException e) { }  
shared = copy + 1;  
System.out.println(Thread.currentThread().getName() + ": " +shared);}}  
Answer  
T1 :1, T2: 2  
T2: 1, T1: 2

//Q6-c)Among the possible outputs of part(a) and (b) which ones would you consider as thread safe?  
Outputs of (b) are thread safe.  
//Q7 The following code is an attempt to utilize wait() and notify() to synchronize threads T1 and T2 in such a way that T2 always runs before T1, thus always guaranteeing the following output:  
T2: 1  
T1: 2  
However, there are bugs in the code that are not letting achieve the above result. Fix the code to guarantee the above output.  
public class ThreadSafety implements Runnable{  
int shared = 0;  
public static void main(String[] args) {  
ThreadSafety ts = new ThreadSafety();  
Thread t1 = new Thread(ts, "T1");  
t1.start();  
Thread t2 = new Thread(ts, "T2");  
t2.start();  
}  
public void run() {  
synchronized(this) {  
if(Thread.currentThread().getName().contains("T2")) {  
try {  
this.wait();  
} catch (InterruptedException e) { }  
}  
int copy = shared;  
try {  
Thread.sleep((int)(Math.random() \* 10000));  
} catch (InterruptedException e) { }  
shared = copy + 1;  
System.out.println(Thread.currentThread().getName() + ": " +shared);}}}  
Answer  
public class ThreadSafety implements Runnable{  
 int shared = 0;  
public static void main(String[] args) {  
ThreadSafety ts = new ThreadSafety();  
Thread t1 = new Thread(ts, "T1");  
t1.start();  
Thread t2 = new Thread(ts, "T2");  
t2.start();  
}  
public void run() {  
synchronized(this) {  
if(Thread.currentThread().getName().contains("T1") && shared == 0) {  
try {  
this.wait();  
} catch (InterruptedException e) { }}   
int copy = shared;  
try {   
Thread.sleep((int)(Math.random() \* 10000));   
} catch (InterruptedException e) { }  
shared = copy + 1;

System.out.println(Thread.currentThread().getName() + ": " +shared);  
 this.notifyAll();  
}}}