*upload - A

- URL routing (web.xml): The URL path /upload is mapped to the Java class UploadServlet. Any GET /upload or POST /upload request is handled by that servlet.
- GET /upload (listing): The servlet reads the folder C:\tomcat\webapps\upload\images, builds a comma-separated list of existing file names, sets response type to text/plain, sets Content-Length, and returns that list. (Note: the current string-building logic prepends "null," as a quirk.)
- POST /upload (upload): The servlet expects a multipart form field named exactly "File". If the submitted file name is empty, it returns HTTP 302 (Found) and stops. Otherwise it saves the file to ${catalina.base}/webapps/upload/images/<filename> and returns a simple text confirmation.

UploadServlet.java

```java
@MultipartConfig // allow parsing multipart/form-data (file uploads)
public class UploadServlet extends HttpServlet {

   // ROUTING NOTE: /upload → UploadServlet (configured in web.xml)

   // doGet: return a comma-separated list of filenames in C:\tomcat\webapps\upload\images as text/plain
   @Override
   protected void doGet(HttpServletRequest req, HttpServletResponse res)
           throws ServletException, IOException {

       // source directory to list for this app
       String dirPath = "C:\\tomcat\\webapps\\upload\\images";

       // build the listing string (original logic yields "null,..." prefix—kept as-is)
       String list = getListing(dirPath);

       // plain text response with length set
       res.setContentType("text/plain");
       res.setContentLength(list ≠ null ? list.length() : 0);
       res.getWriter().print(list ≠ null ? list : "");
   }

   // doPost: accept uploaded file from form field "File", save under ${catalina.base}/webapps/upload/images, respond text
   @Override
   protected void doPost(HttpServletRequest req, HttpServletResponse res)
           throws ServletException, IOException {

       // get the uploaded part; field name is case-sensitive ("File")
       Part filePart = req.getPart("File");

       // extract safe client filename (strip any path)
       String submitted = (filePart ≠ null) ? filePart.getSubmittedFileName() : null;
       String name = (submitted ≠ null) ? Paths.get(submitted).getFileName().toString() : "";

       // if no filename, current behavior: return 302 and stop (note: unusual; 400 is typical)
       if (name.isEmpty()) {
           res.setStatus(HttpServletResponse.SC_FOUND); // 302
           return;
       }

       // resolve ${catalina.base}/webapps/upload/images
       String base = System.getProperty("catalina.base");
       File imagesDir = new File(base + File.separator + "webapps"
               + File.separator + "upload"
               + File.separator + "images");
       imagesDir.mkdirs(); // ensure path exists

       // destination file
       File dest = new File(imagesDir, name);

       // copy upload stream to disk
       try (InputStream in = filePart.getInputStream();
            OutputStream out = new FileOutputStream(dest)) {
           in.transferTo(out);
       }

       // simple OK response
       res.setContentType("text/plain");
       res.getWriter().print("Uploaded: " + name);
   }

   // getListing: return comma-separated file names in 'path' (keeps original "null," prefix quirk)
   private String getListing(String path) {
       File dir = new File(path);
       String list = null;           // starts null → first concat yields "null,<file>"
       String[] files = dir.list();  // may be null if dir missing/unreadable
       if (files ≠ null) {
           for (String f : files) {
               list = list + "," + f; // append with comma
           }
       }
       return list; // caller writes this as text/plain
   }
}
```

---

*trivia - A

LoginServlet.java

```java
// Purpose: GET shows login form; POST authenticates against DB,
//          on success creates session (USER_ID) and redirects to /main.
public class LoginServlet extends HttpServlet {

   // GET /login → render simple login form (username + password)
   @Override
   protected void doGet(HttpServletRequest req, HttpServletResponse res)
           throws ServletException, IOException {

       res.setContentType("text/html");            // HTML response
       PrintWriter out = res.getWriter();

       // Minimal login page (plain form posts back to /login via POST)
       out.println("<!DOCTYPE html><html><head><title>Login</title></head><body>");
       out.println("<h2>Login</h2>");
       out.println("<form method='POST' action='login'>");
       out.println("User: <input type='text' name='user_id' required>");
       out.println("Password: <input type='password' name='password' required>");
       out.println("<button type='submit'>Login</button>");
       out.println("</form>");
       out.println("</body></html>");
   }

   // POST /login → read creds, check DB, set session, redirect to /main
   @Override
   protected void doPost(HttpServletRequest req, HttpServletResponse res)
           throws ServletException, IOException {

       // Read submitted credentials
       String user = req.getParameter("user_id");     // username from form
       String pass = req.getParameter("password");    // password from form

       // DB handles
       Connection con = null;
       Statement  st  = null;
       ResultSet  rs  = null;

       try {
           // Load JDBC driver (must be on classpath)
           Class.forName("oracle.jdbc.OracleDriver");

           // Open connection (course/demo settings)
           con = DriverManager.getConnection(
                   "jdbc:oracle:thin:@localhost:2030:XE", // DB URL
                   "system",                              // DB user
                   "123456");                             // DB password

           // Simple credential check (demo): scan accounts table
           st = con.createStatement();
           rs = st.executeQuery("SELECT username, password FROM accounts");

           boolean ok = false;
           while (rs.next()) {
               String u = rs.getString("username");
               String p = rs.getString("password");
               if (u ≠ null && p ≠ null && u.equals(user) && p.equals(pass)) {
                   ok = true;
                   break;
               }
           }

           if (ok) {
               // Create or reuse session; remember who logged in
               HttpSession session = req.getSession(true);
               session.setAttribute("USER_ID", user);

               // Redirect to main menu (new request; POST body ends here)
               res.sendRedirect("main");
               return;
           }

           // If no match: show simple failure page (reference code often had no explicit message)
           res.setContentType("text/html");
           PrintWriter out = res.getWriter();
           out.println("<!DOCTYPE html><html><body>");
           out.println("<p><b>Login failed.</b></p>");
           out.println("<a href='login'>Try again</a>");
           out.println("</body></html>");

       } catch (ClassNotFoundException e) {
           // Driver missing / classpath issue
           res.setContentType("text/plain");
           res.getWriter().println("DB driver not found: " + e.getMessage());
       } catch (SQLException e) {
           // Connection/query error (bad port/creds/table)
           res.setContentType("text/plain");
           res.getWriter().println("SQL error: " + e.getMessage());
       } finally {
           // Always close JDBC resources
```

```
            try { if (rs ≠ null) rs.close(); } catch (Exception ignore) {}
            try { if (st ≠ null) st.close(); } catch (Exception ignore) {}
            try { if (con ≠ null) con.close(); } catch (Exception ignore) {}
        }

        // Note: some reference code unconditionally set USER_ID and redirected (dev shortcut).
        // If you need that behavior, uncomment below:
        // HttpSession s = req.getSession(true);
        // s.setAttribute("USER_ID", user);
        // res.sendRedirect("main");
    }
}
```

MainServlet.java

```
// Purpose: show main menu after login (session gate) and route to Upload/Play/Logout.
public class MainServlet extends HttpServlet {

    // GET /main → if not logged in, redirect to /login; else render menu
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {

        // Session gate: require USER_ID to view main page
        HttpSession session = req.getSession(false);                // don't create new
        String userId = (session ≠ null) ? (String) session.getAttribute("USER_ID") : null;
        if (userId == null) {                                       // not logged in
            res.sendRedirect("login");                              // bounce to login
            return;
        }

        res.setContentType("text/html");                            // HTML response
        PrintWriter out = res.getWriter();

        // Simple menu UI; form posts back to /main with a button value
        out.println("<!DOCTYPE html><html><head><title>Main</title></head><body>");
        out.println("<h3>Logged in as: " + userId + "</h3>");
        out.println("<form method='POST' action='main'>");
        out.println("<input type='submit' name='click' value='UPLOAD' />");
        out.println("<input type='submit' name='click' value='GALLERY' />"); // label used in ref code
        out.println("<input type='submit' name='click' value='LOGOUT' />");
        out.println("</form>");
        out.println("</body></html>");
    }

    // POST /main → read which button was clicked and redirect accordingly
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {

        String click = req.getParameter("click");                   // which button?

        if (click == null) {                                        // no selection → refresh main
            res.sendRedirect("main");
            return;
        }

        // Route: each choice triggers a redirect (new request to target servlet)
        switch (click) {
            case "UPLOAD":
                res.sendRedirect("upload");                         // go to upload page
                break;
            case "PLAY":                                            // some refs use PLAY internally
            case "GALLERY":                                         // UI label maps to PLAY behavior
                res.sendRedirect("play");                           // go to video/gallery page
                break;
            case "LOGOUT":
                res.sendRedirect("logout");                         // end session in LogoutServlet
                break;
            default:
                res.sendRedirect("main");                           // unknown → stay on main
        }
    }
}
```

FileUploadServlet.java

```
@MultipartConfig // enable multipart/form-data parsing so request.getPart( ... ) works
public class FileUploadServlet extends HttpServlet {

    // GET /upload → render upload form (file + question + date); must include enctype for file upload
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {

        res.setContentType("text/html");
        PrintWriter out = res.getWriter();

        // Minimal form; action posts back to /upload; enctype required for file bytes
        out.println("<!DOCTYPE html><html><head><title>Upload Trivia</title></head><body>");
        out.println("<h3>Upload Trivia</h3>");
        out.println("<form method='POST' action='upload' enctype='multipart/form-data'>");
        out.println("  <div>Image: <input type='file' name='FileName' accept='image/*' required></div>");
        out.println("  <div>Question: <input type='text' name='Question' placeholder='e.g., Who is pictured?'></div>");
        out.println("  <div>UploadDate: <input type='date' name='UploadDate'></div>");
        out.println("  <button type='submit'>Save</button>");
        out.println("</form>");
        out.println("</body></html>");
    }

    // POST /upload → read file Part + fields, insert into DB (BLOB), then select + preview as Base64 <img>
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse res)
            throws ServletException, IOException {

        // Read form parts/params; field names must match the form exactly
        Part filePart = req.getPart("FileName");                    // uploaded binary
        String question = req.getParameter("Question");             // optional text
        String uploadDate = req.getParameter("UploadDate");         // optional date (yyyy-MM-dd)

        // Provide defaults if user left fields blank (matches reference behavior)
        if (question == null || question.isBlank()) question = "Sample Question";     // default question
        if (uploadDate == null || uploadDate.isBlank()) uploadDate = LocalDate.now().toString(); // default date

        // Guard: if no file was actually sent, return a simple message (reference code relies on @MultipartConfig)
        if (filePart == null || filePart.getSize() == 0) {
            res.setContentType("text/html");
            res.getWriter().println("<html><body><p><b>No file selected.</b></p><a href='upload'>Back</a></body></html>");
            return;
        }

        // JDBC handles
        Connection con = null;
        PreparedStatement ps = null;
        Statement st = null;
        ResultSet rs = null;

        // Generate a UUID primary key; DB stores as RAW(16), so convert to 16 bytes
        UUID id = UUID.randomUUID();
        byte[] idBytes = asBytes(id);                               // helper below

        try (InputStream in = filePart.getInputStream()) {          // stream file content once

            // Load driver and connect (demo credentials/port per reference)
            Class.forName("oracle.jdbc.OracleDriver");              // driver must be on classpath
            con = DriverManager.getConnection(
                    "jdbc:oracle:thin:@localhost:49732:XE",         // DB URL (upload service)
                    "system",                                       // DB user
                    "oracle1");                                     // DB password

            // Insert into trivias: (ID RAW(16)), Question VARCHAR2, ContentPath (e.g., mime), Content BLOB
            String sql = "INSERT INTO trivias (id, question, contentpath, content) VALUES (?, ?, ?, ?)";
            ps = con.prepareStatement(sql);
            ps.setBytes(1, idBytes);                                // RAW(16) UUID
            ps.setString(2, question);                              // question text
            ps.setString(3, "image/jpeg");                          // mime type stored in ContentPath
            ps.setBinaryStream(4, in, (int) filePart.getSize());    // BLOB from upload stream
            ps.executeUpdate();

            // Read back one row to preview (simple first-row approach per reference)
            st = con.createStatement();
            rs = st.executeQuery("SELECT id, question, contentpath, content FROM trivias");
            if (rs.next()) {
                // Extract BLOB bytes for inline display
                InputStream blob = rs.getBinaryStream("content");
                byte[] data = blob.readAllBytes();
                String base64 = Base64.getEncoder().encodeToString(data); // encode for data URL

                // Render preview page with inline Base64 <img>
                res.setContentType("text/html");
                PrintWriter out = res.getWriter();
                out.println("<!DOCTYPE html><html><head><title>Upload Preview</title></head><body>");
                out.println("<h3>Saved:</h3>");
                out.println("<p><b>Question:</b> " + escape(question) + "</p>");      // escape minimal
                out.println("<p><b>Date:</b> " + uploadDate + "</p>");
                out.println("<img alt='preview' src='" + base64 + "' style='max-width:640px;'>");
                out.println("<div><a href='main'>Main</a> | <a href='upload'>Upload another</a></div>");
                out.println("</body></html>");
                return;                                             // done
            }

            // If nothing selected (unlikely after insert), just bounce back to main
            res.sendRedirect("main");

        } catch (ClassNotFoundException e) {
            res.setContentType("text/plain");
            res.getWriter().println("DB driver not found: " + e.getMessage());        // driver/classpath issue
        } catch (SQLException e) {
```

```
            res.setContentType("text/plain");
            res.getWriter().println("SQL error: " + e.getMessage());                    // bad creds/port/table/etc.
        } finally {
            // Close JDBC resources
            try { if (rs ≠ null) rs.close(); } catch (Exception ignore) {}
            try { if (st ≠ null) st.close(); } catch (Exception ignore) {}
            try { if (ps ≠ null) ps.close(); } catch (Exception ignore) {}
            try { if (con ≠ null) con.close(); } catch (Exception ignore) {}
        }
    }

    // Convert UUID to 16-byte array (big-endian); matches RAW(16) storage pattern
    private static byte[] asBytes(UUID u) {
        ByteArrayOutputStream out = new ByteArrayOutputStream(16);
        DataOutputStream dos = new DataOutputStream(out);
        try {
            dos.writeLong(u.getMostSignificantBits());
            dos.writeLong(u.getLeastSignificantBits());
            dos.flush();
        } catch (IOException ignored) {}
        return out.toByteArray();
    }

    // Minimal HTML escape for preview text
    private static String escape(String s) {
        if (s == null) return "";
        return s.replace("&", "&amp;").replace("<", "&lt;").replace(">", "&gt;");
    }
}
```

PlayServlet.java

```
// Purpose: GET shows a page that embeds a YouTube video based on DB data (trivias.contentpath).
public class PlayServlet extends HttpServlet {

  // GET /play → query DB for a record and embed YouTube iframe using its contentpath as the video ID
  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
          throws ServletException, IOException {

     res.setContentType("text/html");                    // HTML response
     PrintWriter out = res.getWriter();

     Connection con = null;
     Statement st = null;
     ResultSet rs = null;

     String videoId = null;                              // will hold contentpath (YouTube id)
     String question = null;                             // trivia question (optional to display)

     try {
        // Load driver and connect (per reference config)
        Class.forName("oracle.jdbc.OracleDriver");
        con = DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:3020:XE", // Play servlet DB port
                "system",
                "123456"); // adjust if your reference uses a different password

        // NOTE: Reference code selects id, question, contentpath
        // but later tries to read "video_link" (bug). Use "contentpath" as video id.
        st = con.createStatement();
        rs = st.executeQuery("SELECT id, question, contentpath FROM trivias");

        if (rs.next()) {
           question = rs.getString("question");
           // BUG IN SOME VERSIONS: rs.getString("video_link"); // wrong column
           videoId = rs.getString("contentpath");    // correct column per SELECT
        }

     } catch (ClassNotFoundException e) {
        out.println("<pre>DB driver not found: " + e.getMessage() + "</pre>");
     } catch (SQLException e) {
        out.println("<pre>SQL error: " + e.getMessage() + "</pre>");
     } finally {
        // close JDBC resources
        try { if (rs ≠ null) rs.close(); } catch (Exception ignore) {}
        try { if (st ≠ null) st.close(); } catch (Exception ignore) {}
        try { if (con ≠ null) con.close(); } catch (Exception ignore) {}
     }

     // Build page even if videoId is null (shows a simple message)
     out.println("<!DOCTYPE html><html><head><title>Play Trivia</title></head><body>");
     out.println("<h3>Trivia Player</h3>");

     if (question ≠ null) {
        out.println("<p><b>Question:</b> " + escape(question) + "</p>");
     }

     // If we have a video id, embed it; else show a fallback note
     if (videoId ≠ null && !videoId.isBlank()) {
        // YouTube iframe; start/end/mute/autoplay parameters match reference style
        String src = "https://www.youtube.com/embed/" + videoId + "?autoplay=1&mute=1&start=62&end=162";
        out.println("<iframe width='640' height='360' "
                + "src='" + src + "' "
                + "title='YouTube video player' frameborder='0' "
                + "allow='accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture' "
                + "allowfullscreen></iframe>");
     } else {
        out.println("<p><b>No video available.</b> (Check that 'contentpath' has a YouTube ID.)</p>");
     }

     // Simple navigation buttons: Prev/Next would normally alter which row you select
     out.println("<form method='GET' action='play' style='margin-top:12px;'>");
     out.println("  <button type='submit' name='nav' value='prev'>Prev</button>");
     out.println("  <button type='submit' name='nav' value='next'>Next</button>");
     out.println("  <a href='main'>Main</a>");
     out.println("</form>");

     out.println("</body></html>");
  }

  // Minimal HTML escaper for safe text
  private static String escape(String s) {
     if (s == null) return "";
     return s.replace("&", "&amp;").replace("<", "&lt;").replace(">", "&gt;");
  }
}
```

LogoutServlet.java

```
// Purpose: GET shows a page that embeds a YouTube video based on DB data (trivias.contentpath).
public class PlayServlet extends HttpServlet {

  // GET /play → query DB for a record and embed YouTube iframe using its contentpath as the video ID
  @Override
  protected void doGet(HttpServletRequest req, HttpServletResponse res)
          throws ServletException, IOException {

     res.setContentType("text/html");                    // HTML response
     PrintWriter out = res.getWriter();

     Connection con = null;
     Statement st = null;
     ResultSet rs = null;

     String videoId = null;                              // will hold contentpath (YouTube id)
     String question = null;                             // trivia question (optional to display)

     try {
        // Load driver and connect (per reference config)
        Class.forName("oracle.jdbc.OracleDriver");
        con = DriverManager.getConnection(
                "jdbc:oracle:thin:@localhost:3020:XE", // Play servlet DB port
                "system",
                "123456"); // adjust if your reference uses a different password

        // NOTE: Reference code selects id, question, contentpath
        // but later tries to read "video_link" (bug). Use "contentpath" as video id.
        st = con.createStatement();
        rs = st.executeQuery("SELECT id, question, contentpath FROM trivias");

        if (rs.next()) {
           question = rs.getString("question");
           // BUG IN SOME VERSIONS: rs.getString("video_link"); // wrong column
           videoId = rs.getString("contentpath");    // correct column per SELECT
        }

     } catch (ClassNotFoundException e) {
        out.println("<pre>DB driver not found: " + e.getMessage() + "</pre>");
     } catch (SQLException e) {
        out.println("<pre>SQL error: " + e.getMessage() + "</pre>");
     } finally {
        // close JDBC resources
        try { if (rs ≠ null) rs.close(); } catch (Exception ignore) {}
        try { if (st ≠ null) st.close(); } catch (Exception ignore) {}
        try { if (con ≠ null) con.close(); } catch (Exception ignore) {}
     }

     // Build page even if videoId is null (shows a simple message)
     out.println("<!DOCTYPE html><html><head><title>Play Trivia</title></head><body>");
     out.println("<h3>Trivia Player</h3>");

     if (question ≠ null) {
        out.println("<p><b>Question:</b> " + escape(question) + "</p>");
     }

     // If we have a video id, embed it; else show a fallback note
     if (videoId ≠ null && !videoId.isBlank()) {
```

```
        // YouTube iframe; start/end/mute/autoplay parameters match reference style
        String src = "https://www.youtube.com/embed/" + videoId + "?autoplay=1&mute=1&start=62&end=162";
        out.println("<iframe width='640' height='360' "
                + "src='" + src + "' "
                + "title='YouTube video player' frameborder='0' "
                + "allow='accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture' "
                + "allowfullscreen></iframe>");
    } else {
        out.println("<p><b>No video available.</b> (Check that 'contentpath' has a YouTube ID.)</p>");
    }

    // Simple navigation buttons: Prev/Next would normally alter which row you select
    out.println("<form method='GET' action='play' style='margin-top:12px;'>");
    out.println("  <button type='submit' name='nav' value='prev'>Prev</button>");
    out.println("  <button type='submit' name='nav' value='next'>Next</button>");
    out.println("  <a href='main'>Main</a>");
    out.println("</form>");

    out.println("</body></html>");
  }

  // Minimal HTML escaper for safe text
  private static String escape(String s) {
    if (s == null) return "";
    return s.replace("&", "&amp;").replace("<", "&lt;").replace(">", "&gt;");
  }
}
```

```
*triviaapi - A
TRIVIAAPI — RUNDOWN
1) What this is
    •   A tiny API-style servlet pair:
        ○   Account: “log in” endpoint that just sets a session attribute (USER_ID). No DB, no HTML, no redirect.
        ○   Trivia: REST-ish servlet with @MultipartConfig that:
            ▪   GET: requires a logged-in session; lists images folder and returns a minimal body.
            ▪   POST: accepts a multipart file upload and writes it to the app’s /images folder.
            ▪   PUT/DELETE: stubs that log and return 200.
2) Flow you should picture
    1.  Client POST /account with user_id (+ maybe password) → server creates/gets session, sets USER_ID, returns 200.
    2.  Client GET /trivia → if session is valid: 200 with a simple response (current code prints an <img> tag string for the first file in /images); if not logged in: 302 → /login.
    3.  Client POST /trivia (multipart) with file part name = fileName (+ optional caption, date) → file saved under ${catalina.base}/webapps/triviaapi/images/ → 200. If filename empty → 302 → /upload.
    4.  PUT /trivia, DELETE /trivia → placeholders; log and 200.
3) Account servlet (login shim)
    •   POST:
        ○   Reads user_id and password (password isn’t validated).
        ○   request.getSession(true); session.setAttribute("USER_ID", user_id).
        ○   response.setStatus(200). No JSON/HTML body. No redirects. No DB.
    •   GET: no UI (left empty in reference).
Exam cues: This is a session setter, not real auth. Meant to prime the session so Trivia GET passes the login check.
4) Trivia servlet (core API)
    •   Annotated @MultipartConfig so request.getPart( ... ) works.
GET
    •   Session gate: uses a helper to check getSession(false) and isRequestedSessionIdValid(). If not logged in → 302 + redirect to login.
    •   On success:
        ○   Content-Type: text/plain (even though it prints an HTML-ish <img> line — that mismatch is intentional/harmless for the exercise).
        ○   Looks in c:\tomcat\webapps\triviaapi\images, grabs the first filename, writes one line like:
            <img src='images/<name>' alt='Image'/>
        ○   Sets 200 OK (even if the folder is empty).
POST
    •   Fields:
        ○   File part fileName (must match exactly).
        ○   Optional caption, date.
    •   Behavior:
        ○   If submitted filename is empty → 302 to upload and return.
        ○   If date empty → default "2020-10-10".
        ○   If caption empty → default "No caption".
        ○   Writes file to: ${catalina.base}/webapps/triviaapi/images/<fileName>.
        ○   200 OK (no body).
PUT / DELETE
    •   Log path/params; set 200. No actual update/delete logic.
Helper: isLoggedIn(req)
    •   Returns false if there’s no session or the requested session id is invalid; otherwise true.

Account.java
// Purpose: Minimal login endpoint for triviaapi.
// POST sets session attribute USER_ID; no DB check/HTML/redirect; returns 200.
public class Account extends HttpServlet {

   // GET /account → no UI in this reference (intentionally empty)
   @Override
   public void doGet(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
       // intentionally left blank (could render a small form if needed)
   }

   // POST /account → accept user_id/password and establish session
   @Override
   public void doPost(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {

       String username = request.getParameter("user_id"); // submitted username
       String password = request.getParameter("password"); // submitted password (unused here)

       // Create (or reuse) session and remember the user (key checked by other servlets)
       HttpSession session = request.getSession(true);
       session.setAttribute("USER_ID", username);

       System.out.println("Logged in as:" + username);   // server-side trace only
       response.setStatus(200);                          // plain 200 OK (no redirect/body)
   }
}
```

```
Trivia.java
@MultipartConfig // allow multipart/form-data uploads (used in doPost)
public class Trivia extends HttpServlet {

   // GET /trivia (or mapped path) → require session; if logged in, list images and return a minimal body
   @Override
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {

       HttpSession session = request.getSession(false);          // do not create a session here
       boolean isLoggedIn = isLoggedIn(request);                 // validate JSESSIONID + session presence
       if (!isLoggedIn) {
           response.setStatus(302);                              // redirect if not logged in
           response.sendRedirect("login");
           return;
       }

       response.setContentType("text/plain");                    // sends plain text (not JSON/HTML)
       response.setCharacterEncoding("UTF-8");

       // Diagnostics: path/params (server logs only; students can ignore on exam)
       System.out.println(request.getPathInfo());
       System.out.println(request.getParameterMap());

       // List files in the API's images directory (Tomcat webapps folder for this app)
       File dir = new File("c:\\tomcat\\webapps\\triviaapi\\images");
       String[] fileList = dir.list();

       // NOTE: They intended a JSON array (commented code); current output is an <img> string.
       if (fileList != null && fileList.length > 0) {
           String imgTag = "<img src = 'images/" + fileList[0] + "'alt='Image'/>"; // very minimal
           PrintWriter out = response.getWriter();
           out.println(imgTag);                                  // body is just a single line
       }
       response.setStatus(200);                                  // OK (even if folder empty)
   }

   // PUT /trivia → placeholder that logs request info and returns 200
   @Override
   protected void doPut(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
       System.out.println(request.getPathInfo());                // debug: path
       System.out.println(request.getParameterMap());            // debug: params
       response.setStatus(200);
   }

   // DELETE /trivia → placeholder that logs request info and returns 200
   @Override
   protected void doDelete(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {
       System.out.println(request.getPathInfo());                // debug: path
       System.out.println(request.getParameterMap());            // debug: params
       response.setStatus(200);
   }

   // POST /trivia → accept file upload (field "fileName"), plus optional "caption" and "date"; save to
webapps/triviaapi/images
   @Override
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
           throws ServletException, IOException {

       System.out.println("In Do Post");                         // server-side trace

       Part filePart = request.getPart("fileName");              // uploaded file part (field name must match)
       String captionName = request.getParameter("caption");     // optional text
       String formDate = request.getParameter("date");           // optional date (yyyy-MM-dd)
       String fileName = filePart.getSubmittedFileName();        // original filename
```

```
        // If no file chosen → redirect to /upload (matches reference behavior)
        if (fileName.equals("")) {
            response.setStatus(302);
            response.sendRedirect("upload");
            return;
        }

        // Provide defaults if user left fields blank
        if (formDate.equals("")) formDate = "2020-10-10";          // default date
        if (captionName.equals("")) captionName = "No caption";    // default caption

        // Save the file under Tomcat's webapps folder for this app, so it's directly accessible under /triviaapi/images/...
        filePart.write(System.getProperty("catalina.base") + "/webapps/triviaapi/images/" + fileName);

        response.setStatus(200);                                    // success (no body/redirect)
    }

    // Helper: treat user as logged in only if we have a valid session id and session exists
    private boolean isLoggedIn(HttpServletRequest req) {
        HttpSession session = req.getSession(false);
        if (session == null || !req.isRequestedSessionIdValid()) {  // missing or invalid JSESSIONID
            return false;
        } else {
            return true;
        }
    }
}
```

---

1) <form> (HTML Form tag)
What it does: Wraps controls (inputs, selects, buttons) and defines how form data is submitted.
Key attributes
- action: URL to submit to (e.g., /api/login).
- method: get (query string) or post (request body).
- enctype: encoding (e.g., application/x-www-form-urlencoded (default), multipart/form-data for file uploads, text/plain).
- target: where to open the response (_self, _blank, etc.).
- autocomplete: on|off.
- novalidate: bypass built-in validation when present.

Notes
- Pressing Enter in a text input submits the *nearest* form.
- You can intercept with JS: event.preventDefault().

Example
```
<form id="loginForm" action="/api/login" method="post" novalidate>
  <input name="email" type="email" required />
  <input name="password" type="password" required minlength="8" />
  <button type="submit">Sign in</button>
</form>
```

---

2) <button> (HTML Button tag)
Types
- type="submit": submits the nearest <form>.
- type="button": generic click button (no submit).
- type="reset": resets form controls to initial values.

Tips
- The default inside a form is type="submit" — be explicit to avoid surprises.
- Can contain HTML (icons, spans), unlike <input type="submit">.

Example
```
<button type="button" id="openHelp"><span>❓</span> Help</button>
<button type="reset">Clear</button>
<button type="submit">Save</button>
```

---

3) <input type="submit">
What it does: A submit control rendered as a button.
Key attributes
- value: button label.
- name/value: if set, gets included in the form data (useful when you have multiple submit buttons and want to know which one was used).

Compare vs <button type="submit">
- <input> cannot contain HTML (text only).
- <button> can hold rich content but has slightly different default styling across browsers.

Example
```
<input type="submit" value="Upload" name="whichSubmit" />
```

---

4) document.getElementById()
What it does: Returns the element with the matching id (or null).
Signature: HTMLElement|null getElementById(string id)
Notes
- id should be unique.
- Very fast, no CSS selectors needed; if you need selectors, use querySelector.

Example
```
<div id="status"></div>
<script>
  const statusEl = document.getElementById('status');
  if (statusEl) statusEl.textContent = 'Ready ';
</script>
```

---

5) window.location.href
What it does: Gets/sets the current URL.

```
Read current URL
console.log(window.location.href);
Navigate (adds to history):
window.location.href = '/dashboard';
Other related methods
```
- location.assign(url): same as setting href (back works).
- location.replace(url): navigates without creating a history entry.
- location.reload(): reloads the page (optionally true to force reload).

---

6) XMLHttpRequest (XHR)
What it is: Legacy JS API for HTTP requests. Works everywhere. (Modern alternative: fetch.)
Creating & basic flow
```
const xhr = new XMLHttpRequest();
xhr.open('POST', '/api/login', true); // true = async
xhr.setRequestHeader('Content-Type', 'application/json');
xhr.onreadystatechange = () => {
  // fires 0→4 times as readyState changes
  if (xhr.readyState === 4) {
    if (xhr.status >= 200 && xhr.status < 300) {
      console.log('OK', xhr.responseText);
    } else {
      console.error('Error', xhr.status, xhr.responseText);
    }
  }
};
xhr.send(JSON.stringify({ email, password }));
```

Properties (commonly used)
- readyState: 0=UNSENT, 1=OPENED, 2=HEADERS_RECEIVED, 3=LOADING, 4=DONE
- status: HTTP status code (e.g., 200, 404)
- statusText: HTTP status text (e.g., "OK")
- response: response body, type depends on responseType
- responseText: response as string
- responseXML: response parsed as XML (if XML)
- responseURL: final URL after redirects
- responseType: "" (string), "json", "blob", "document", "arraybuffer"
- timeout: ms until request times out
- withCredentials: include cookies/credentials for CORS
- upload: an XMLHttpRequestUpload object for tracking upload progress

Methods
- open(method, url, async=true, user?, password?)
- send(body?)
- abort()
- setRequestHeader(name, value)
  getResponseHeader(name)
- getAllResponseHeaders()
- overrideMimeType(mime)

Events (handlers or addEventListener)
- readystatechange: fires on each readyState change
- load: completes successfully (status available)
  error: network-level failure
- abort: aborted
- timeout: exceeded timeout
- progress: download progress (on xhr)
- loadstart, loadend: lifecycle
- Upload events (on xhr.upload): progress, load, error, abort, timeout, loadstart, loadend

Progress example
```
const xhr = new XMLHttpRequest();
xhr.open('POST', '/upload');

xhr.upload.onprogress = (e) => {
  if (e.lengthComputable) {
    const pct = Math.round((e.loaded / e.total) * 100);
    console.log('Uploaded', pct + '%');
  }
};

xhr.onload = () => console.log('Done', xhr.status);
xhr.onerror = () => console.error('Network error');

const data = new FormData();
data.append('file', fileInput.files[0]);
xhr.send(data);
```

DaoPattern.java - B
```
// Create the base class with abstract serialize method that derived classes will need to implement
abstract class Game {
  String GameType;                  // consider 'private final' with a getter for encapsulation
  public Game(String gameType) { this.GameType = gameType; } // trusts caller; could validate non-null
  abstract String serialize();      // contract for a stable wire/text format
}

//Quiz class inherits from abstract Game class
class Quiz extends Game {
  int Id;                           // consider 'private' + getters/setters to enforce invariants
  String Question;                  // consider trimming/validating input
  String ContentPath;               // might want to validate URL-ish content
```

```
  public Quiz() { super("Quiz"); } //this is default constructor
  public Quiz(String constructorParams) { // this constructor parses the constructorParams to set the object state
     super("Quiz");
     // NOTE: this simplistic parser breaks if values contain ',' or '='; real code should escape or use JSON/CSV libs
     String[] keyvaluePairs = constructorParams.split(",");
     for(int i = 0; i < keyvaluePairs.length; i++) {
        String[] keyvaluePair = keyvaluePairs[i].split("="); // assumes "key=value"; no bounds check
        switch (keyvaluePair[0]) {
           case "Id": this.Id = Integer.parseInt(keyvaluePair[1]); break;     // may throw NumberFormatException
           case "Question": this.Question = keyvaluePair[1]; break;           // could decode if encoded
           case "ContentPath": this.ContentPath = keyvaluePair[1];break;      // consider URL decoding
        }
     }
  }
  void setId(int id) { this.Id = id; }                    // consider validating id > 0
  void setQuestion (String question) { this.Question = question; } // consider null/blank checks
  void setContentPath (String contentPath) { this.ContentPath = contentPath; } // consider normalization
  //Quiz's implementation of the serialize() method
  String serialize() { return "Id="+this.Id+",Question="+this.Question+",ContentPath="+this.ContentPath; } // values not
escaped
}

//Some other class that also inherits from abstract Game class.
class Puzzle extends Game {
  int Id;                          // same encapsulation considerations as Quiz
  String Name;
  String Details;
  public Puzzle() { super("Puzzle"); }
  public Puzzle(String constructorParams) { super("Puzzle"); } //implementation not provided for now // (would mirror Quiz
parsing)
  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; } // same escaping caveat
}

//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) {
     // consider using enum for type-safety and switch over enum
     if (gameType.equalsIgnoreCase("Quiz")) {
        return new Quiz(constructorParameters);
     } else if (gameType.equalsIgnoreCase("Puzzle")) {
        return new Puzzle(constructorParameters);
     }
     return null; // returns null on unknown type; caller should null-check (or throw IllegalArgumentException)
  }
}

// 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
  // consider adding update/delete methods for full CRUD; also return types or exceptions for error handling
}

//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;                    // not thread-safe; use Collections.synchronizedList if multi-threaded
  public Repository() { gameList = new ArrayList<Game>(); }
  @Override
  public void insert(Game game) { gameList.add(game) ; } // no null check; consider Objects.requireNonNull
  @Override
  public List<Game> select(String gameType, String pattern) {   //note the use of Functional Programming in this method
     Stream<Game> gameStream = gameList.stream();
     // IMPORTANT: '==' compares references for String; use .equals( ... ) for content comparison.
     // Also consider null-safety: gameType could be null.
     Stream<Game> newGameStream = gameStream.filter(s -> s.GameType == gameType && s.serialize().contains(pattern));
     // better: .filter(s -> Objects.equals(s.GameType, gameType) && s.serialize().contains(pattern))
     List<Game> selectedGames = new ArrayList<Game>();
     GameFactory gameFactory = new GameFactory();
     // This re-materializes new instances from serialize(); acts like mapping DB rows back to entities.
     newGameStream.forEach(s -> selectedGames.add(gameFactory.createGame(s.GameType, s.serialize())));
     return selectedGames; // note: may contain nulls if factory returns null on unknown type
  }
}
public class DaoPattern
{
  public static void main(String[] args) {
     //repository object should be created in the constructor of the Servlet;
     IRepository repository = new Repository(); // in real servlet, prefer a long-lived field (avoid per-request
instantiation)
     //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"); // typo "Governal"
kept as-is
     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"); // relies on serialize() containing "Id=2"
     for (Game game : selectedGames) {
        System.out.println(game.serialize()); // prints "key=value,... " string; consumer must parse if needed
     }
  }
}
```

```
*ConsoleApp - B
UploadClient.java
import java.io.*;                    // streams for file/socket I/O
import java.net.*;                   // Socket

public class UploadClient {
   public UploadClient() { }         // trivial ctor (no state)

   public String uploadFile() {
       String listing = "";
       try {
           // Connect to server on localhost:8999 (hard-coded endpoint)
           Socket socket = new Socket("localhost", 8999);

           // Wrap server input to read line-by-line (server will send text response)
           BufferedReader in = new BufferedReader(
                   new InputStreamReader(socket.getInputStream()));

           OutputStream out = socket.getOutputStream(); // raw byte stream to server

           // Read entire file into memory; consider streaming for large files
           FileInputStream fis = new FileInputStream("AndroidLogo.png"); // assumes CWD contains file
           byte[] bytes = fis.readAllBytes();             // Java 9+ convenience; may be big

           out.write(bytes);                              // send file bytes
           socket.shutdownOutput();                       // signal "done sending" to server (half-close)

           fis.close();                                   // release file handle

           System.out.println("Came this far\n");         // simple progress log (stdout)

           String filename = "";
           // Read server response lines until EOF; server likely sends file name or status
           while ((filename = in.readLine()) != null) {
               listing += filename;                       // accumulate without newline; append '\n' if needed
           }

           socket.shutdownInput();                        // half-close input (optional; close socket also closes streams)
       } catch (Exception e) {
           System.err.println(e);                         // coarse error reporting; consider logging and rethrow
       }
       return listing;                                    // return server's response text
   }
}
```

```
Activity.java
import java.io.*;                     // not used directly here (kept as-is)

public class Activity {
  private String dirName = null;      // field present but unused; potential future config (e.g., upload folder)

  public static void main(String[] args) throws IOException {
     new Activity().onCreate();       // emulate Android-style lifecycle in a console app
  }

  public Activity() {                 // default ctor; could accept config (server host/port, file path)
  }

  public void onCreate() {
     // Kick off upload and print server response; in real app, handle null/empty and errors
     System.out.println(new UploadClient().uploadFile());
  }
}
```

*UploadServer

BIG PICTURE
This is a tiny “servlet-like” server. It:
1. Listens on TCP port 8999
2. Accepts a client connection and spawns a thread per connection
3. Wraps the socket streams as Request/Response objects
   Invokes UploadServlet#doPost(req, res) to do the app work
4. Sends the servlet’s output back over the socket, then closes the connection

Important: This is not a real HTTP server. There’s no HTTP request line/header parsing and no HTTP response headers. It expects raw bytes from the client.

FILES & ROLES
- UploadServer.java
  - Starts a ServerSocket(8999)
    Infinite accept loop → new UploadServerThread(client).start()
- UploadServerThread.java (per-connection worker)
  - Gets InputStream/OutputStream from the socket
  - Builds HttpServletRequest with the socket’s InputStream
  - Creates a ByteArrayOutputStream, wraps it in HttpServletResponse
    Instantiates UploadServlet and calls doPost(req, res)
  - Writes the response buffer back to the client socket and closes the socket
- HttpServlet.java
  - Minimal base “framework” class with empty doGet(...) and doPost(...) hooks
    UploadServerThread calls doPost(...)
- HttpServletRequest.java
  - Holds the raw InputStream from the socket
  - getInputStream() returns the incoming bytes (no URL/headers/params)
- HttpServletResponse.java
  - Holds an OutputStream (here, a ByteArrayOutputStream buffer)
  - getOutputStream() is where the servlet writes its reply
- UploadServlet.java (application logic)
  - doPost(...) reads all bytes from the request InputStream
  - Saves them to <currentMillis>.png
  - Writes a plain-text directory listing (filenames in “.”) to the response

REQUEST LIFECYCLE (STEP-BY-STEP)
1. Client connects to port 8999.
2. UploadServer.accept() returns a Socket, starts UploadServerThread.
3. Thread wraps:
   - req = new HttpServletRequest(socket.getInputStream())
   - res = new HttpServletResponse(new ByteArrayOutputStream())
4. Thread calls new UploadServlet().doPost(req, res).
5. Inside doPost:
   - Reads bytes from req.getInputStream() until EOF
   - Writes bytes to a file named <epochMillis>.png
   - Prints each filename in the current directory to res.getOutputStream() (one per line)
6. Thread takes the in-memory response bytes and writes them to the socket’s output.
7. Thread closes the socket. Done.

WHAT A CLIENT MUST SEND
- Raw bytes, then close the connection.
- Because there’s no HTTP parsing, if you send a real HTTP POST, HTTP headers will be saved into the .png file → corrupt output.

QUICK LOCAL TEST (EXAMPLES)
- Send a raw file using nc (netcat):
  - Linux/macOS: nc 127.0.0.1 8999 < somefile.bin
  - Windows (Ncat): ncat 127.0.0.1 8999 < somefile.bin
- You should receive back a plain-text listing of files in the server’s working directory.
- Check that a new file named like 1699999999999.png was created on the server.

CONCURRENCY & THREAD SAFETY
- One thread per connection.
- UploadServlet uses method-local variables → no shared mutable state in the servlet.
- Shared resource: filesystem. Filename uses milliseconds; extremely rare collision if two uploads land in the exact same millisecond.

LIMITATIONS / PITFALLS
- Not HTTP: no status line, no headers, no content-type.
- Unbounded reads: a huge upload can consume a lot of memory/disk.
- Blocking I/O: a client that never closes the connection can keep a thread busy.
- Hard-coded “.png” extension regardless of actual content.
- Reading with a 1-byte buffer in the reference is inefficient (works, just slow).

COMMON “WHAT IF THE EXAM CHANGES X?” POINTS
- Change doPost → doGet call in the thread:
  - Base doGet is a no-op unless the servlet overrides it → no output.
- Use a bigger buffer (e.g., byte[8192]):
  - Same behavior, much faster.
    Add HTTP headers to the response:
  - Some clients will parse it better (e.g., prepend HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\n\r\n) but you still aren’t parsing the request.
- Parse real multipart/form-data:
  - Would require implementing HTTP header parsing and boundary handling (not in this stack).
- Stream response directly to socket.getOutputStream() instead of BAOS:
  - Works but removes the “buffer then flush” pattern; you’d write as you go.

UploadServer.java
```java
import java.net.*;  // networking APIs (ServerSocket, Socket)
import java.io.*;   // basic I/O (IOException)

public class UploadServer {
    public static void main(String[] args) throws IOException {  // entry point of the server process
        ServerSocket serverSocket = null;                         // handle for the listening TCP socket
        try {
            serverSocket = new ServerSocket(8999);                // bind and listen on TCP port 8999
        } catch (IOException e) {
            System.err.println("Could not listen on port: 8999."); // if the port is in use or blocked, fail fast
            System.exit(-1);                                       // terminate the process (no server means nothing to do)
        }

        // accept-loop: handle each incoming connection on its own thread (concurrency via thread-per-connection)
        while (true) {
            Socket client = serverSocket.accept();                // BLOCKS until a client connects; returns a Socket
            new UploadServerThread(client).start();               // spin up a worker thread to process this client
        }
    }
}
```

UploadServerThread.java
```java
import java.net.*;      // networking APIs (Socket)
import java.io.*;       // basic I/O (InputStream/OutputStream, ByteArrayOutputStream)
import java.time.Clock; // unused here (UploadServlet uses Clock for filenames)

public class UploadServerThread extends Thread {           // per-connection worker thread
  private Socket socket = null;                            // the client connection handled by this thread

  public UploadServerThread(Socket socket) {
     super("DirServerThread");                             // give the thread a name (helps in debugging)
     this.socket = socket;                                 // remember the client socket for run()
  }

  public void run() {                                      // thread entry point
     try {
        InputStream in = socket.getInputStream();          // bytes coming from the client
        HttpServletRequest req = new HttpServletRequest(in);  // wrap client input as a tiny Request object

        OutputStream baos = new ByteArrayOutputStream();   // buffer to capture servlet output in memory
        HttpServletResponse res = new HttpServletResponse(baos); // wrap buffer as Response object

        HttpServlet httpServlet = new UploadServlet();     // instantiate the application servlet
        httpServlet.doPost(req, res);                      // run app logic: read upload → save file → build reply

        OutputStream out = socket.getOutputStream();       // bytes going back to the client
        out.write(((ByteArrayOutputStream) baos).toByteArray()); // send the buffered response back over the socket

        socket.close();                                    // close the connection (also closes streams)
     } catch (Exception e) { e.printStackTrace(); }        // log any error; thread ends afterward
  }
}
```

HttpServletRequest.java
```java
import java.io.*;                          // InputStream lives here

public class HttpServletRequest {
  private InputStream inputStream = null; // raw bytes coming from the client socket

  public HttpServletRequest(InputStream inputStream) {
     this.inputStream = inputStream;     // store the underlying stream
  }

  public InputStream getInputStream() {  // servlet reads request body from here
     return inputStream;
  }
}
```

HttpServletResponse.java
```java
import java.io.*;                           // OutputStream lives here

public class HttpServletResponse {
  private OutputStream outputStream = null; // where the servlet writes its reply (buffered or direct)

  public HttpServletResponse(OutputStream outputStream) {
     this.outputStream = outputStream;     // store the underlying stream
  }

  public OutputStream getOutputStream() {  // servlet obtains a sink to write response bytes
     return outputStream;
  }
}
```

UploadServlet.java
```java
import java.io.*;                          // streams, files, PrintWriter
import java.time.Clock;                    // clock used to make a timestamped filename

public class UploadServlet extends HttpServlet {
  protected void doPost(HttpServletRequest request, HttpServletResponse response) {
     try {
        InputStream in = request.getInputStream();       // raw bytes from the client (no HTTP parsing)
        ByteArrayOutputStream baos = new ByteArrayOutputStream();  // accumulate the entire upload in memory

        byte[] content = new byte[1];                    // 1-byte buffer (works but is inefficient)
```

```
        int bytesRead = -1;
        while( ( bytesRead = in.read( content ) ) != -1 ) {  // read until client closes stream (EOF == -1)
           baos.write( content, 0, bytesRead );          // append bytes to in-memory buffer
        }

        Clock clock = Clock.systemDefaultZone();         // get system clock
        long milliSeconds = clock.millis();              // current epoch millis → simple unique-ish name

        OutputStream outputStream =
                new FileOutputStream(new File(String.valueOf(milliSeconds) + ".png")); // write to <millis>.png
        baos.writeTo(outputStream);                      // dump the uploaded bytes to the file on disk
        outputStream.close();                            // close file handle

        PrintWriter out = new PrintWriter(response.getOutputStream(), true); // text writer for response body

        File dir = new File(".");                        // current working directory
        String[] chld = dir.list();                      // list its entries (names only)
        for(int i = 0; i < chld.length; i++){
           String fileName = chld[i];
           out.println(fileName + "\n");                 // return a simple plain-text listing (one per line)
           System.out.println(fileName);                 // also log to server stdout
        }
     } catch(Exception ex) {
        System.err.println(ex);                          // basic error logging (no HTTP status returned)
     }
  }
}
```

SimpleThread.java - C

```
import java.io.*;
public class SimpleThread extends Thread {
   public SimpleThread(String str) {        // ctor sets thread name via super
      super(str);
   }

   public void run() {                      // thread entry point (executed after start())
      for (int i = 0; i < 10; i++) {       // do 10 iterations
         System.out.println(i + " " + getName()); // prints counter + thread name
         try {
            sleep((long)(Math.random() * 1000)); // pause 0–999 ms; sleep() keeps the monitor if held
         } catch (InterruptedException e) {}      // interrupt ignored; best practice: restore interrupt flag
      }
      System.out.println("DONE! " + getName());    // signal completion
   }
}
// Notes: extending Thread is fine for demos; in apps prefer `implements Runnable` for flexibility (composition).
// If a caller needs to wait for completion, they should call `thread.join()` from outside.
```

ThreadSafety.java

```
public class ThreadSafety implements Runnable{
   int shared = 0; // shared mutable state; races avoided only because all accesses are inside synchronized(this)
   public static void main(String[] args) throws InterruptedException
   {
       ThreadSafety ts = new ThreadSafety(); // both threads share the same runnable → same monitor 'this' and same
'shared'
       Thread t1 = new Thread(ts, "T1");
       t1.start(); // T1 may or may not reach wait() before T2 runs (ordering not guaranteed)
       Thread t2 = new Thread(ts, "T2");
       t2.start(); // potential "lost notification" if notify() happens before T1 begins waiting
   }
   public void run() {
       synchronized(this)
       {
           boolean t2done = false; // flag to indicate if T2 has completed
           // NOTE: this flag is LOCAL to each thread invocation (not shared); it only guards the while-loop structure for
T1
           // If you actually need cross-thread coordination, use a shared flag (e.g., volatile boolean done) or
CountDownLatch.

           if (Thread.currentThread().getName().contains("T1"))
           {
               // T1 path: wait until T2 signals; relies on notify() occurring AFTER T1 is waiting.
               // If T2 enters run() first and calls notify() before T1 calls wait(), T1 can block forever (lost notify).
               while (!t2done)
               {
                   try
                   {
                       this.wait(); // releases 'this' monitor and waits; must be in a loop
                       t2done = true; // sets the flag to true once T2 is done
                       // This sets a local predicate; no shared state is checked here, so correctness depends on timing.
                   }
                   catch (InterruptedException e) { } // swallow → thread continues; consider restoring interrupt
               }
           }
           this.notify(); // notify T1 that T2 is done, so t1 can run
           // If current thread is T2, this may wake T1 as intended.
           // If current thread is T1 (after waking), this notify likely wakes nobody (benign).
           // Risk: if T2 executed and notified BEFORE T1 started waiting, T1 will wait forever (nobody left to notify).
           int copy = shared; // read under lock → consistent
           try
           {
               Thread.sleep((int)(Math.random() * 10000)); // sleep while holding the monitor; reduces concurrency and can
delay the other thread
               // Best practice: minimize time inside synchronized blocks; move sleeps/long work outside when possible.
           }
           catch (InterruptedException e) { } // swallow; consider handling or re-set interrupt:
Thread.currentThread().interrupt();
           shared = copy + 1; // write under lock → no data race
           System.out.println(Thread.currentThread().getName() + ": " +shared); // likely prints "T2: 1" then "T1: 2" when
timing is favorable
       }
   }
}
```

CubbyHole.java

```
public class CubbyHole {
   private int contents;                    // single-slot buffer (one value at a time)
   private boolean available = false;       // condition flag: true = slot full

   public synchronized int get(int who) { // monitor method (guards contents/available)
      while (available == false) {       // condition loop to avoid spurious wakeups
         try {
            wait();                      // releases monitor; waits for put()
         } catch (InterruptedException e) { }
      }
      available = false;                    // consume the item → slot becomes empty
      System.out.println("Consumer " + who + " got: " + contents);
      notifyAll();                          // wake producers waiting in put()
      return contents;                      // hand back the consumed value
   }

   public synchronized void put(int who, int value) { // monitor method for producers
      while (available == true) {        // wait while full (avoid overwrite)
         try {
            wait();                      // releases monitor; waits for get()
         } catch (InterruptedException e) { }
      }
      contents = value;                     // write new value
      available = true;                     // mark as full
      System.out.println("Producer " + who + " put: " + contents);
      notifyAll();                          // wake consumers waiting in get()
   }
}
```

Producer.java

```
public class Producer extends Thread {
   private CubbyHole cubbyhole;          // shared monitor
   private int number;                   // producer id (for logging)

   public Producer(CubbyHole c, int number) {
      cubbyhole = c;
      this.number = number;
   }

   public void run() {
      for (int i = 0; i < 10; i++) {    // produce 10 items
         cubbyhole.put(number, i);      // blocks if buffer full until consumer gets
         try {
            sleep((int)(Math.random() * 100)); // jitter to vary interleaving
         } catch (InterruptedException e) { }
      }
   }
}
```

Consumer.java

```
public class Consumer extends Thread {
   private CubbyHole cubbyhole;          // shared monitor
   private int number;                   // consumer id (for logging)

   public Consumer(CubbyHole c, int number) {
      cubbyhole = c;
      this.number = number;
   }

   public void run() {
      int value = 0;
      for (int i = 0; i < 10; i++) {    // consume 10 items
         value = cubbyhole.get(number); // blocks if buffer empty until producer puts
      }
   }
}
```

ProducerConsumerTest.java

```
public class ProducerConsumerTest {
   public static void main(String[] args) {
      CubbyHole c1 = new CubbyHole();           // each buffer is independent
```

```
      Producer p11 = new Producer(c1, 1);
      Consumer c11 = new Consumer(c1, 1);

      CubbyHole c2 = new CubbyHole();   // second, separate buffer
      Producer p12 = new Producer(c2, 2);
      Consumer c12 = new Consumer(c2, 2);

      p11.start();                              // start threads (non-deterministic order)
      c11.start();
      p12.start();
      c12.start();
   }
}
```

SemaphoreTest.java

```
import java.util.concurrent.Semaphore;
public class SemaphoreTest {
  // max 4 people
  static Semaphore semaphore = new Semaphore(4);  // counting semaphore (permits=4)

  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();                   // may block until a permit is available
           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);              // simulates work while holding the permit
              }
           } finally {
              // calling release() after a successful acquire()
              System.out.println(name + " : releasing lock.");
              semaphore.release();                // returns permit; wakes a waiter if any
              System.out.println(name + " : available Semaphore permits now: "
                    + semaphore.availablePermits());
           }
        } catch (InterruptedException e) {
           e.printStackTrace();                   // interrupted while waiting or sleeping
        }
     }
  }

  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");       // these extra threads will wait
     t5.start();
     MyATMThread t6 = new MyATMThread("F");       // until a permit is released
     t6.start();
  }
}
```

MultiThreading.pdf

1. Implementing Runnable (Alternative to extends Thread)

Cleaner design using composition instead of inheritance.

```
class Clock implements Runnable {
   private volatile Thread clockThread;
   public void start() {
      if (clockThread == null) {
         clockThread = new Thread(this, "Clock");
         clockThread.start();
      }
   }
   public void run() {
      while (Thread.currentThread() == clockThread) {
         System.out.println(new java.util.Date());
         try { Thread.sleep(1000); } catch (InterruptedException e) {}
      }
   }
}
```

Key ideas:

- Prefer implements Runnable for flexibility.
- A Runnable can be passed to multiple Thread instances.
- Use volatile for shared flags to safely stop loops.

2. Thread Priority & yield()

```
Thread t1 = new Thread(() -> { while (true) System.out.println("A"); });
Thread t2 = new Thread(() -> { while (true) System.out.println("B"); });
t1.setPriority(Thread.MAX_PRIORITY);
t2.setPriority(Thread.MIN_PRIORITY);
t1.start(); t2.start();
```

- Priorities range from 1 (MIN_PRIORITY) to 10 (MAX_PRIORITY).
- yield() temporarily pauses the thread to let equal-priority threads run.
- Use for efficiency only — not correctness (scheduler behavior isn't guaranteed).

3. Timed wait() Variants (beyond the indefinite version)

You only used wait() with no timeout; Java also provides:

```
wait(long timeout);          // ms
wait(long timeout, int ns); // ms + ns
```

Use cases:

- To prevent infinite blocking.
- Difference: wait() releases the monitor; sleep() doesn't.
  wait() can be woken early by notify(); sleep() cannot.

4. Starvation & Deadlock (Definitions)

- Starvation: Thread never gets CPU/resource due to scheduling or priority.
- Deadlock: Two or more threads each wait on a lock held by another.
  Prevent with consistent lock ordering or using tryLock(timeout) on ReentrantLock.

5. Explicit Locks & Condition Variables

Use when you need finer control than synchronized.

```
import java.util.concurrent.locks.*;
class CubbyHole2 {
   private int contents; private boolean available = false;
   private final Lock lock = new ReentrantLock();
   private final Condition cond = lock.newCondition();

   public void put(int who, int value) throws InterruptedException {
      lock.lock();
      try {
         while (available) cond.await();
         contents = value; available = true;
         System.out.println("Producer " + who + " put: " + contents);
         cond.signalAll();
      } finally { lock.unlock(); }
   }

   public int get(int who) throws InterruptedException {
      lock.lock();
      try {
         while (!available) cond.await();
         available = false;
         System.out.println("Consumer " + who + " got: " + contents);
         cond.signalAll();
         return contents;
      } finally { lock.unlock(); }
   }
}
```

- ReentrantLock allows explicit control over acquiring/releasing locks.
- Condition.await() and signalAll() replace wait()/notifyAll().
- Always use try/finally to ensure lock release.

6. Timer & TimerTask Classes

You already have AnnoyingBeep.java and Reminder.java, but the key point from the PDF:

- These classes manage threads internally for periodic or delayed execution.
- Use Timer.schedule(TimerTask task, long delay, long period) for repeated tasks.

Assignment1