دانشکده فنی و مهندسی

برنامه نویسی پیشرفته – دستور العمل پروژه نهایی

این پروژه نهایی جهت تثبیت، جمع‌بندی و به‌کارگیری عملی مفاهیم پیشرفته برنامه‌نویسی شی‌گرا مانند: **وراثت، رابط‌ها، کلاس‌های داخلی، چندنخی و طراحی رابط گرافیکی** طراحی شده است.

خروجی این پروژه باید حتماً به صورت رابط گرافیکی(GUI) باشد و برنامه‌های تحت کنسول پذیرفته نخواهند شد. زبان پیاده‌سازی ترجیحا Java است (با تمرکز بر کتابخانه Swing یا JavaFX) با این حال، استفاده از C++ یا Python مجاز است، مشروط بر اینکه تمام مفاهیم شیءگرایی ذکر شده در صورت پروژه دقیقاً پیاده‌سازی شوند.

لطفا برای پروژه یک ساختار مرتب و منظم (common، server، client) ایجاد کنید. فایل ارسالی با فرمت zip و با عنوان نام‌دانشجو_کددانشجویی باشد. imirmobin@gmail.com

---

مستندات جامع پروژه نهایی: سیستم فروشگاه آنلاین(Mini DigiKala)

۱) مقدمه و تعریف مسئله

در دنیای واقعی، نرم‌افزارهای بزرگ (Enterprise) هرگز روی یک کامپیوتر اجرا نمی‌شوند. آن‌ها از معماری **کلاینت–سرور** استفاده می‌کنند. دیجی‌کالا، آمازون یا اسنپ را تصور کنید: دیتابیس و منطق اصلی روی سرورهای قدرتمند است و هزاران کاربر (کلاینت) از طریق اپلیکیشن یا سایت به آن وصل می‌شوند.

هدف این پروژه شبیه‌سازی کوچک‌شده‌ای از این اکوسیستم است. شما قرار است هم نقش **Backend Developer** (توسعه سرور و دیتابیس) و هم نقش **Frontend Developer** (طراحی رابط گرافیکی دسکتاپ) را بازی کنید.

۲) معماری سیستم

این سیستم از سه جزء اصلی تشکیل شده است که باید با هم "حرف بزنند":

- **سرور:** این برنامه همیشه روشن است. به دیتابیس متصل است و منتظر می‌ماند تا کسی درخواستی بفرستد (مثلاً: "لیست موبایل‌ها را بده"). سرور باید بتواند همزمان پاسخ چندین نفر را بدهد(Multi-threading).
- **کلاینت:** یک برنامه گرافیکی (Swing GUI) که کاربر با آن کار می‌کند. این برنامه هیچ مغزی ندارد! یعنی اگر کاربر دکمه "خرید" را زد، کلاینت فقط یک پیام به سرور می‌فرستد و منتظر تایید می‌ماند.
- **پایگاه داده:** محل ذخیره دائمی محصولات، کاربران و سفارش‌ها.

۳) سناریوهای کاربردی

در این سیستم دو نوع بازیگر داریم: **مشتری** و **مدیر**.

۱-۳) سناریوی مشتری

وقتی یک کاربر عادی وارد برنامه می‌شود، باید بتواند کارهای زیر را انجام دهد:

۱-۱-۳) ویترین گردی:

- کاربر باید لیستی از محصولات را به صورت کارت‌های گرافیکی (عکس + قیمت + نام) ببیند.
- **دسته‌بندی درختی:** در کنار صفحه، باید دسته‌بندی‌ها به صورت تو در تو (درختی) باشند. مثلاً وقتی روی "کالای دیجیتال" کلیک کرد، زیرمجموعه‌های "موبایل" و "لپ‌تاپ" باز شوند. (این بخش نیازمند الگوریتم بازگشتی است).
- **جستجو و فیلتر:** کاربر بتواند بر اساس نام یا قیمت کالاها را فیلتر کند.

۲-۱-۳) جزئیات محصول:

- با کلیک روی هر کالا، صفحه جزئیات باز شود.
- اگر کالا **فیزیکی** بود، وزن و هزینه ارسال نمایش داده شود.
- اگر کالا **مجازی** (مثل لایسنس نرم‌افزار) بود، حجم فایل نمایش داده شود.
- **نظرات:** کاربر بتواند نظرات دیگران را ببیند و خودش نظر ثبت کند.

۳-۱-۳) سبد خرید و پرداخت:

- کاربر کالاها را به سبد اضافه می‌کند.
- سیستم باید قیمت کل را محاسبه کند. (دقت کنید: هزینه ارسال فقط برای کالاهای فیزیکی اضافه می‌شود).
- با زدن دکمه "تکمیل خرید"، موجودی انبار در سرور کم می‌شود و یک فاکتور (Order) در دیتابیس ثبت می‌شود.

۳-۲) سناریوی مدیر

اگر کسی با نام کاربری و رمز مدیر (admin) وارد شد، ظاهر برنامه باید تغییر کند و تب **"مدیریت"** ظاهر شود:

۳-۲-۱) مدیریت موجودی:

- مدیر می‌تواند کالای جدید تعریف کند.
- باید مشخص کند کالا فیزیکی است یا مجازی؟ (کلاس‌های PhysicalProduct و DigitalProduct)
- تغییر قیمت یا موجودی کالاها.

۳-۲-۲) داشبورد وضعیت:

- لیست تمام کاربرانی که الان آنلاین هستند.
- لیست سفارشاتی که ثبت شده ولی هنوز ارسال نشده‌اند.

۴) تشریح موجودیت‌ها

برای پیاده‌سازی این سیستم، شما با داده‌های زیر سر و کار دارید:

۴-۱) محصولات (Products)

ما یک مفهوم کلی به نام "محصول" داریم، اما رفتار محصولات متفاوت است:

Physical Product: وقتی خریده شد، باید آدرس پستی کاربر گرفته شود و هزینه پست روی فاکتور بیاید.

Digital Product: وقتی خریده شد، لینک دانلود به کاربر نمایش داده می‌شود و هزینه پستی ندارد.

چالش شما: استفاده از Polymorphism(چندریختی) تا وقتی دکمه "محاسبه فاکتور" زده شد، برنامه خودش بفهمد برای کدام محصول هزینه پست بگیرد و برای کدام نگیرد.

۲-۴) **تعاملات (Interactions)** – مبحث جنریک و بازگشتی

پیام‌های سرور: سرور ممکن است "لیست کالا" بفرستد، یا "نتیجه لاگین" یا "پیام خطا". چطور همه این‌ها را در یک ساختار بفرستیم؟ با استفاده از Generic(Response<T>)

کامنت‌های تو در تو: علی نظر می‌دهد "خوبه". رضا زیر نظر علی می‌نویسد "موافقم". سارا زیر نظر رضا می‌نویسد "چرا؟". نمایش این ساختار درختی نیاز به **Recursion** دارد.

۵) پروتکل ارتباطی

این بخش "زبان مشترک" کلاینت و سرور شماست. شما باید متدهای جاوا داشته باشید:

عملیات	متد در جاوا (Interface)	توضیح
ورود	`login(user, pass)`	بررسی نام کاربری و بازگرداندن اطلاعات کاربر
لیست کالا	`getAllProducts()`	دریافت آرایه‌ای از تمام محصولات
ثبت کالا	`addProduct(product)`	(فقط ادمین) افزودن یک شیء محصول به دیتابیس
خرید	`submitOrder(cart)`	نهایی کردن خرید و کسر موجودی

ساختار پیشنهادی پروژه:

می‌توانید از این ساختار برای نوشتن پروژه استفاده کنید.

```
StudentNumber_Name_FinalProject/    <-- پوشه اصلی
│
├── database.sql                    <-- (فایل خروجی دیتابیس)
│
├── src/                            <-- (سورس کدها)
│   ├── client/
│   │   ├── LoginFrame.java
│   │   └── ...
│   ├── server/
│   │   ├── ServerMain.java
│   │   └── ...
│   └── common/
│       ├── Product.java
│       └── ...
│
└── (سایر فایل‌های پروژه مانند عکس‌ها یا کتابخانه‌ها)
```

پکیج common (مشترکات پروژه):

این پوشه حاوی کلاس‌هایی است که هم "سرور" باید آن‌ها را بشناسد (تا در دیتابیس ذخیره کند) و هم "کلاینت" (تا بتواند نشان دهد). فایل‌های این پوشه هیچ "منطق پیچیده" یا "کد دیتابیسی" ندارند؛ فقط مدل‌سازی داده‌ها هستند.

نام فایل	نوع (Class/Interface)	وظیفه و توضیحات
Product.java	abstract class	کلاس والد محصولات. فیلدها: نام، قیمت، آیدی، توضیحات.
PhysicalProduct.java	class	فرزند کلاس بالا. فیلد اضافه: وزن، هزینه ارسال.
DigitalProduct.java	class	فرزند کلاس بالا. فیلد اضافه: لینک دانلود.
User.java	class	مشخصات کاربر (نام کاربری، پسورد، نقش).
Response.java	class <T>	**(جنریک)** ظرفی برای ارسال جواب از سرور به کلاینت. شامل: وضعیت (موفق/ناموفق) و داده اصلی.
ShopService.java	interface	**(بسیار مهم)** لیست کارهایی که سرور می‌تواند انجام دهد (Login, Search, Buy). کلاینت فقط این را می‌بیند.
Enums.java	enum	مقادیر ثابت مثل UserRole (ADMIN, CUSTOMER) یا Category.

پکیج server (سرور):

حاوی تمام کدهای مربوط به **دیتابیس، منطق تجاری** و **مدیریت شبکه** است. کلاینت هیچکدام از فایل‌های این پوشه را ندارد.

نام فایل	وظیفه و توضیحات
ServerMain.java	کلاس اصلی اجرایی. یک ServerSocket می‌سازد و در یک حلقه while(true) منتظر اتصال کلاینت می‌ماند.
ClientHandler.java	کلاسی که Runnable را پیاده‌سازی می‌کند (برای تردینگ). وقتی یک کلاینت وصل شد، سرور یک نمونه از این کلاس می‌سازد تا اختصاصی با آن کلاینت حرف بزند.
ShopServiceImpl.java	این کلاس، اینترفیس ShopService (از پوشه common) را implements می‌کند. کد واقعی لاگین کردن یا جستجو در اینجا نوشته می‌شود.
DatabaseHandler.java	تمام کدهای JDBC (مثل Connection, PreparedStatement) فقط و فقط در این فایل هستند. هیچ جای دیگر پروژه نباید SQL ببینیم.

پکیج **client** (کلاینت):

این بخش حاوی ارسال درخواست و گرافیک برنامه(GUI) است. در دیتابیس نقشی ندارد.

نام فایل	وظیفه و توضیحات
ClientMain.java	نقطه شروع برنامه کلاینت. فرم لاگین را باز می‌کند.
NetworkClient.java	کلاسی که سوکت (Socket) سمت کلاینت را مدیریت می‌کند. درخواست‌ها را می‌فرستد و منتظر Response می‌ماند.
LoginFrame.java	(Swing) پنجره ورود و ثبت نام.

نام فایل	وظیفه و توضیحات
`MainFrame.java`	(Swing) پنجره اصلی که تب‌بندی شده (فروشگاه، سبد خرید، مدیریت).
`ProductPanel.java`	(Swing) یک پنل کوچک مستطیلی که عکس و قیمت **یک** محصول را نشان می‌دهد (برای نمایش در لیست).
`Cart.java`	یک کلاس ساده (احتمالاً جنریک) برای نگهداری موقت کالاهایی که کاربر انتخاب کرده (قبل از پرداخت نهایی).

چک لیست فنی

۱ .پکیج common

- کلاس **Product (Abstract)**:
  - فیلدها: id, name, price, stock, description
  - متد انتزاعی: abstract double calculateFinalPrice()
- متدtoString() : Override شود.
- کلاس **PhysicalProduct (extends Product)**:
  - فیلدها: weight, shippingCost
  - پیاده‌سازی: calculateFinalPrice جمع قیمت + هزینه پست.
- کلاس **DigitalProduct (extends Product)**:
  - فیلدها: downloadLink, fileSize
  - پیاده‌سازی: calculateFinalPrice فقط قیمت پایه.
- کلاس **Response<T> (Generic)**:
  - فیلدها: Status (Enum), String message, T data
- اینترفیس **ShopService**:
  - Response<User> login(String user, String pass)
  - Response<List<Product>> getAllProducts()
  - Response<String> addProduct(Product p)
  - Response<String> submitOrder(Order o)

۲ .پکیج server

- کلاس **DatabaseHandler**:
  - مدیریت درایور JDBC و Connection
  - اجرای کوئری‌های(SELECT, INSERT, UPDATE) SQL

- کلاس **ServerMain:**
  - ایجاد ServerSocket روی پورت ۸۰۸۰.
  - حلقه بی‌نهایت while(true) برای accept کردن کلاینت.
- کلاس **ClientHandler (implements Runnable)**:
  - مدیریت سوکت اختصاصی هر کاربر.
  - اجرای متد run برای خواندن و نوشتن در استریم‌ها.

۳ .پکیج client

- کلاس **MainFrame**:
  - استفاده از JTabbedPaneبا سه تب: ۱. فروشگاه، ۲. سبد خرید، ۳. پنل ادمین.
- کلاس **ProductPanel (extends JPanel)**:
  - نمایش گرید (Grid) محصولات.
  - هر محصول شامل: تصویر، نام، قیمت، دکمه خرید.
- **رخدادها(Events)**:
  - تمامی ActionListener ها باید به صورت **Anonymous Inner Class** باشند.
- **بازگشتی(Recursion)** :
  - استفاده از JTree برای نمایش دسته‌بندی‌ها.
  - متد پر کردن نودهای درخت باید **بازگشتی** باشد.

۴ .دیتابیس(Tables)

- **جدول Users**: ستون‌های .id, username, password, role
- **جدول Products:**: ستون‌های .id, name, price, stock, type
- **جدولOrders**: ستون‌های .id, user_id, total_amount

۵ .مدیریت خطا(Exceptions)

- تعریف کلاس‌هایOutOfStockException, InvalidLoginException
- ارسال آبجکت Response با وضعیت ERROR در صورت بروز خطا.
- نمایش خطا در کلاینت با JOptionPane.showMessageDialog