ESTRUCTURA:

```
BoletaMaster/
│
├── src/
│   ├── logica/
│   │     Usuario.java
│   │     Cliente.java
│   │     Organizador.java
│   │     Administrador.java
│   │     Evento.java
│   │     Venue.java
│   │     Localidad.java
│   │     Tiquete.java
│   │     Transaccion.java
│   │     Transferencia.java
│   │     Oferta.java
│   │     ConfiguracionCobros.java
│   │     ReporteFinanciero.java
│   │
│   ├── excepciones/
│   │     SaldoInsuficienteException.java
│   │     UsuarioNoAutenticadoException.java
│   │     TiqueteNoTransferibleException.java
│   │     EventoNoAprobadoException.java
│   │
│   └── presentacion/
│         DemoMain.java   ← se corre desde aquí
│
└── (archivos .txt para persistencia futura)
```

Perfecto — te doy **todo listo**: estructura de proyecto sugerida, **archivos Java completos** (con excepciones personalizadas), y un `DemoMain` que ejecuta **escenarios automáticos** y muestra en consola los resultados en un estilo natural (como lo haría un estudiante pero con calidad para obtener buena nota). Sigue las instrucciones de ejecución al final.

---

1) Estructura recomendada del proyecto (Eclipse)

Crea un proyecto Java llamado `BoletaMaster` y dentro de `src` crea estas carpetas (paquetes):

```
src/
 ├─ logica
 ├─ excepciones
 └─ presentacion
```

Pega cada uno de los siguientes archivos Java en el paquete indicado.

---

2) Código Java (pega cada archivo en su paquete correspondiente)

> Todos los archivos están en Java 8+ estándar. Las excepciones son **checked** para forzar manejo y mostrar mensajes claros en consola.

Paquete `excepciones`

`excepciones/SaldoInsuficienteException.java`

```
package excepciones;

public class SaldoInsuficienteException extends Exception {
    public SaldoInsuficienteException(String mensaje) {
        super(mensaje);
    }
}
```

`excepciones/UsuarioNoAutenticadoException.java`

```
package excepciones;

public class UsuarioNoAutenticadoException extends Exception {
    public UsuarioNoAutenticadoException(String mensaje) {
        super(mensaje);
    }
}
```

`excepciones/TiqueteNoTransferibleException.java`

```
package excepciones;

public class TiqueteNoTransferibleException extends Exception {
    public TiqueteNoTransferibleException(String mensaje) {
        super(mensaje);
    }
}
```

`excepciones/EventoNoAprobadoException.java`

```
package excepciones;

public class EventoNoAprobadoException extends Exception {
    public EventoNoAprobadoException(String mensaje) {
        super(mensaje);
    }
}
```

`excepciones/LimiteTiquetesException.java`

```
package excepciones;

public class LimiteTiquetesException extends Exception {
    public LimiteTiquetesException(String mensaje) {
        super(mensaje);
    }
}
```

Paquete `logica` — Usuarios y Paquete

`logica/Usuario.java`

```
package logica;

public abstract class Usuario {
    protected String login;
    protected String passwordHash;
    protected boolean activo = true;

    public Usuario(String login, String password) {
        this.login = login;
        this.passwordHash = hash(password);
    }

    private String hash(String pwd) {
        return Integer.toHexString((pwd == null) ? 0 : pwd.hashCode());
    }

    public boolean autenticar(String password) {
        return activo && hash(password).equals(passwordHash);
    }

    public String getLogin() { return login; }
```

```
    public void desactivar() { activo = false; }
}
```

logica/PaqueteDeluxe.java

```
package logica;

public class PaqueteDeluxe {
    private String beneficios;
    private boolean incluyeTiquetesAdicionales;

    public PaqueteDeluxe(String beneficios, boolean incluyeTiquetesAdicionales) {
        this.beneficios = beneficios;
        this.incluyeTiquetesAdicionales = incluyeTiquetesAdicionales;
    }

    public String getBeneficios() { return beneficios; }
    public boolean incluyeTiquetesAdicionales() { return incluyeTiquetesAdicionales; }
}
```

logica/Cliente.java

```
package logica;

import excepciones.*;
import logica.tickets.*;
import logica.eventos.*;
import java.util.*;

public class Cliente extends Usuario {
    private double saldo;
    private Map<String,Tiquete> tiquetes = new HashMap<>();
    private List<Transaccion> transacciones = new ArrayList<>();
    private static final int MAX_TICKETS_PER_TX = 10;

    public Cliente(String login, String password, double saldoInicial) {
        super(login, password);
        this.saldo = saldoInicial;
    }

    public double getSaldo() { return saldo; }
    public void acreditar(double monto) { saldo += monto; }
    public void debitar(double monto) { saldo -= monto; }

    public Collection<Tiquete> getTiquetes() { return tiquetes.values(); }
```

```
    // Compra (lanza excepciones si falla)
    public Transaccion comprarTiquetes(Evento evento, Localidad localidad, int cantidad,
double precioUnitario)
            throws SaldoInsuficienteException, LimiteTiquetesException,
EventoNoAprobadoException {

        if (!evento.getVenue().isAprobado()) {
            throw new EventoNoAprobadoException("El venue del evento no está aprobado.");
        }
        if (cantidad <= 0) {
            throw new IllegalArgumentException("Cantidad debe ser > 0.");
        }
        if (cantidad > MAX_TICKETS_PER_TX) {
            throw new LimiteTiquetesException("Se supera el límite de tiquetes por transacción:
" + MAX_TICKETS_PER_TX);
        }
        if (localidad.getCapacidadDisponibles() < cantidad) {
            throw new IllegalStateException("No hay disponibilidad suficiente en la localidad.");
        }

        double total = precioUnitario * cantidad;
        if (saldo < total) {
            throw new SaldoInsuficienteException("Saldo insuficiente: saldo=" + saldo + ", total="
+ total);
        }

        Transaccion trx = new Transaccion(total, "SALDO");
        for (int i = 0; i < cantidad; i++) {
            Tiquete t = new Tiquete(localidad, precioUnitario, this);
            trx.agregarTiquete(t);
            tiquetes.put(t.getId(), t);
            localidad.marcarVendido();
        }
        transacciones.add(trx);
        debitar(total);
        return trx;
    }

    // Transferencia
    public Transferencia transferirTiquete(String idTiquete, Cliente destino, String pwd)
            throws UsuarioNoAutenticadoException, TiqueteNoTransferibleException {

        if (!this.autenticar(pwd)) {
            throw new UsuarioNoAutenticadoException("Contraseña incorrecta para usuario " +
login);
        }
        Tiquete t = tiquetes.get(idTiquete);
```

```
        if (t == null) {
            throw new IllegalArgumentException("Tiquete no encontrado: " + idTiquete);
        }
        if (!t.isTransferible()) {
            throw new TiqueteNoTransferibleException("Tiquete no transferible: " + idTiquete);
        }
        if (t.isUsado()) {
            throw new IllegalStateException("Tiquete ya usado: " + idTiquete);
        }

        // realizar transferencia
        tiquetes.remove(idTiquete);
        t.setPropietario(destino);
        destino.recibirTiquete(t);
        Transferencia tr = new Transferencia(this.login, destino.getLogin());
        return tr;
    }

    public void recibirTiquete(Tiquete t) {
        tiquetes.put(t.getId(), t);
    }

    public void imprimirTiquetes() {
        System.out.println("Tiquetes de " + login + ":");
        if (tiquetes.isEmpty()) {
            System.out.println("  (ninguno)");
            return;
        }
        for (Tiquete t : tiquetes.values()) {
            System.out.println("  - " + t);
        }
    }
}
```

`logica/Organizador.java`

```
package logica;

import logica.eventos.*;
import logica.finanzas.ReporteFinanciero;
import java.util.*;

public class Organizador extends Usuario {
    private String nombreEmpresa;
    private List<Evento> eventos = new ArrayList<>();

    public Organizador(String login, String password, String nombreEmpresa) {
```

```
        super(login, password);
        this.nombreEmpresa = nombreEmpresa;
    }

    public Evento crearEvento(String id, String nombre, String fecha, String hora, String tipo,
Venue venue) {
        Evento e = new Evento(id, nombre, fecha, hora, tipo, "pendiente", venue);
        eventos.add(e);
        return e;
    }

    public void generarReporte(ReporteFinanciero r) {
        System.out.println("Organizador " + login + " genera reporte " + r.getIdReporte());
    }
}
```

`logica/Administrador.java`

```
package logica;

import logica.eventos.Venue;
import logica.finanzas.ConfiguracionCobros;

public class Administrador extends Usuario {
    public Administrador(String login, String pwd) { super(login, pwd); }

    public void aprobarVenue(Venue v) {
        v.setAprobado(true);
        System.out.println("[ADMIN] Venue aprobado: " + v.getNombre());
    }

    public void cancelarEvento(logica.eventos.Evento e) {
        e.setEstado("cancelado");
        System.out.println("[ADMIN] Evento cancelado: " + e.getNombre());
    }

    public void configurarCobros(ConfiguracionCobros cfg) {
        System.out.println("[ADMIN] Configuración de cobros aplicada.");
    }
}
```

Paquete `logica.eventos` — eventos, localidad, venue, oferta

`logica/eventos/Venue.java`

```
package logica.eventos;

public class Venue {
    private String id;
    private String nombre;
    private String ubicacion;
    private int capacidad;
    private boolean aprobado = false;

    public Venue(String id, String nombre, String ubicacion, int capacidad) {
        this.id = id; this.nombre = nombre; this.ubicacion = ubicacion; this.capacidad =
capacidad;
    }

    public String getNombre() { return nombre; }
    public int getCapacidad() { return capacidad; }
    public boolean isAprobado() { return aprobado; }
    public void setAprobado(boolean aprobado) { this.aprobado = aprobado; }

    @Override public String toString() { return nombre + " (" + ubicacion + ")"; }
}
```

`logica/eventos/Localidad.java`

```
package logica.eventos;

public class Localidad {
    private String id;
    private String nombre;
    private double precioBase;
    private boolean numerada;
    private int capacidad;
    private int vendidos = 0;

    public Localidad(String id, String nombre, double precioBase, boolean numerada, int
capacidad) {
        this.id = id; this.nombre = nombre; this.precioBase = precioBase; this.numerada =
numerada; this.capacidad = capacidad;
    }

    public int getCapacidadDisponibles() { return capacidad - vendidos; }
    public void marcarVendido() { vendidos++; }
    public int getVendidos() { return vendidos; }
    public double getPrecioBase() { return precioBase; }
    public String getNombre() { return nombre; }

    @Override public String toString() { return nombre + " (precio=" + precioBase + ")"; }
```

```
}
```

logica/eventos/Evento.java

```
package logica.eventos;

import java.util.*;

public class Evento {
    private String id;
    private String nombre;
    private String fecha;
    private String hora;
    private String tipo;
    private String estado;
    private Venue venue;
    private List<Localidad> localidades = new ArrayList<>();

    public Evento(String id, String nombre, String fecha, String hora, String tipo, String
estado, Venue venue) {
        this.id = id; this.nombre = nombre; this.fecha = fecha; this.hora = hora; this.tipo = tipo;
this.estado = estado; this.venue = venue;
    }

    public void asociarLocalidad(Localidad l) { localidades.add(l); }
    public List<Localidad> getLocalidades() { return localidades; }
    public String getNombre() { return nombre; }
    public String getEstado() { return estado; }
    public void setEstado(String estado) { this.estado = estado; }
    public Venue getVenue() { return venue; }
}
```

logica/eventos/Oferta.java

```
package logica.eventos;

public class Oferta {
    private String id;
    private double porcentaje;
    private long inicio;
    private long fin;
    private boolean activa;

    public Oferta(String id, double porcentaje, long inicio, long fin, boolean activa) {
        this.id = id; this.porcentaje = porcentaje; this.inicio = inicio; this.fin = fin; this.activa =
activa;
```

```
    }

    public boolean aplicaAhora() {
        long now = System.currentTimeMillis();
        return activa && now >= inicio && now <= fin;
    }

    public double aplicarDescuento(double precio) {
        if (aplicaAhora()) return precio * (1 - porcentaje/100.0);
        return precio;
    }
}
```

Paquete `logica.tickets` — tiquetes y transacciones

`logica/tickets/Tiquete.java`

```
package logica.tickets;

import logica.eventos.Localidad;
import logica.Cliente;
import java.util.UUID;

public class Tiquete {
    protected String id;
    protected double precioFinal;
    protected String fechaCompra;
    protected String estado;
    protected boolean transferible;
    protected Cliente propietario;
    protected Localidad localidad;
    protected boolean usado = false;

    public Tiquete(Localidad localidad, double precioFinal, Cliente propietario) {
        this.id = UUID.randomUUID().toString();
        this.localidad = localidad;
        this.precioFinal = precioFinal;
        this.fechaCompra = java.time.LocalDateTime.now().toString();
        this.estado = "valido";
        this.transferible = true;
        this.propietario = propietario;
    }

    public String getId() { return id; }
    public boolean isTransferible() { return transferible; }
```

```
    public void setTransferible(boolean t) { transferible = t; }
    public boolean isUsado() { return usado; }
    public void marcarUsado() { usado = true; estado = "usado"; }
    public void setPropietario(Cliente c) { propietario = c; }
    public Cliente getPropietario() { return propietario; }
    public Localidad getLocalidad() { return localidad; }
    public double getPrecioFinal() { return precioFinal; }

    @Override public String toString() {
        return String.format("T[%s] - %s - estado=%s - precio=%.2f", id.substring(0,6),
localidad.getNombre(), estado, precioFinal);
    }
}
```

logica/tickets/TiqueteIndividual.java

```
package logica.tickets;

import logica.eventos.Localidad;
import logica.Cliente;

public class TiqueteIndividual extends Tiquete {
    private String numeroAsiento;
    public TiqueteIndividual(Localidad localidad, double precioFinal, Cliente propietario, String
numeroAsiento) {
        super(localidad, precioFinal, propietario);
        this.numeroAsiento = numeroAsiento;
    }
}
```

logica/tickets/TiqueteMultipleEvento.java

```
package logica.tickets;

import logica.eventos.Localidad;
import logica.Cliente;

public class TiqueteMultipleEvento extends Tiquete {
    private int cantidadEntradas;
    public TiqueteMultipleEvento(Localidad localidad, double precioFinal, Cliente propietario,
int cantidad) {
        super(localidad,precioFinal,propietario);
        this.cantidadEntradas = cantidad;
    }
}
```

logica/tickets/Transaccion.java

```
package logica.tickets;

import java.util.*;
import java.util.UUID;

public class Transaccion {
    private String id;
    private String fecha;
    private String estado;
    private double total;
    private String tipoPago;
    private List<Tiquete> tiquetes = new ArrayList<>();

    public Transaccion(double total, String tipoPago) {
        this.id = UUID.randomUUID().toString();
        this.fecha = java.time.LocalDateTime.now().toString();
        this.estado = "completada";
        this.total = total;
        this.tipoPago = tipoPago;
    }

    public void agregarTiquete(Tiquete t) { tiquetes.add(t); }
    public String getId() { return id; }
    public List<Tiquete> getTiquetes() { return tiquetes; }
}
```

logica/tickets/Transferencia.java

```
package logica.tickets;

import java.util.UUID;

public class Transferencia {
    private String id;
    private String fecha;
    private String estado;
    private String fromLogin;
    private String toLogin;

    public Transferencia(String fromLogin, String toLogin) {
        this.id = UUID.randomUUID().toString();
        this.fecha = java.time.LocalDateTime.now().toString();
        this.estado = "completada";
        this.fromLogin = fromLogin; this.toLogin = toLogin;
    }
```

```
    public String getId() { return id; }
}
```

Paquete `logica.finanzas` — configuración y reporte

`logica/finanzas/ConfiguracionCobros.java`

```
package logica.finanzas;

public class ConfiguracionCobros {
    private double pctMusical;
    private double pctDeportivo;
    private double pctCultural;
    private double pctReligioso;
    private double cargoEmision;

    public ConfiguracionCobros(double m, double d, double c, double r, double em) {
        this.pctMusical = m; this.pctDeportivo = d; this.pctCultural = c; this.pctReligioso = r;
this.cargoEmision = em;
    }

    public double calcularComision(double precioBase, String tipoEvento) {
        double pct = 0;
        switch (tipoEvento.toLowerCase()) {
            case "musical": pct = pctMusical; break;
            case "deportivo": pct = pctDeportivo; break;
            case "cultural": pct = pctCultural; break;
            case "religioso": pct = pctReligioso; break;
            default: pct = pctCultural; break;
        }
        return precioBase * pct / 100.0 + cargoEmision;
    }
}
```

`logica/finanzas/ReporteFinanciero.java`

```
package logica.finanzas;

import java.util.UUID;

public class ReporteFinanciero {
    private String idReporte;
    private String tipo;
    private String fechaGeneracion;
```

```
    private String datos;

    public ReporteFinanciero(String tipo, String datos) {
        this.idReporte = UUID.randomUUID().toString();
        this.tipo = tipo; this.datos = datos;
        this.fechaGeneracion = java.time.LocalDateTime.now().toString();
    }

    public String getIdReporte() { return idReporte; }
}
```

Paquete `presentacion` — DemoMain (ejecución automática)

`presentacion/DemoMain.java`

```
package presentacion;

import logica.*;
import logica.eventos.*;
import logica.tickets.*;
import logica.finanzas.*;
import excepciones.*;

public class DemoMain {
    public static void main(String[] args) {
        System.out.println("=== BoletaMaster - Demo automático (Entrega 2) ===\n");

        // Crear actores
        Cliente juan = new Cliente("juan", "pass123", 200.0);
        Cliente maria = new Cliente("maria", "pass456", 50.0);
        Organizador org = new Organizador("organizador1", "orgpass", "MusicCo");
        Administrador admin = new Administrador("admin", "adminpass");

        // Crear venue y aprobarlo
        Venue estadio = new Venue("V001", "Estadio Central", "Av. Principal", 1000);
        System.out.println("[INFO] Organizador sugiere un venue: " + estadio);
        admin.aprobarVenue(estadio);

        // Crear evento
        Evento concierto = org.crearEvento("E100", "Concierto Rock", "2025-11-01", "20:00",
"musical", estadio);
        Localidad platea = new Localidad("L1","Platea",30.0,false,100);
        Localidad vip = new Localidad("L2","VIP",60.0,true,10);
        concierto.asociarLocalidad(platea);
        concierto.asociarLocalidad(vip);
```

```
        System.out.println("\n== Historia 1: Compra de tiquetes ==");
        try {
            Transaccion t = juan.comprarTiquetes(concierto, platea, 2, platea.getPrecioBase());
            System.out.println("[OK] " + juan.getLogin() + " compró 2 tiquetes. Transacción: " +
t.getId());
        } catch (SaldoInsuficienteException | LimiteTiquetesException |
EventoNoAprobadoException e) {
            System.out.println("[ERROR] Compra fallida: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("[ERROR] Excepción inesperada en compra: " + e.getMessage());
        }

        juan.imprimirTiquetes();
        System.out.println(juan.getLogin() + " - Saldo restante: " + juan.getSaldo());

        System.out.println("\n== Historia 2: Transferencia de tiquete ==");
        try {
            String idToTransfer = juan.getTiquetes().stream().findFirst().map(t ->
t.getId()).orElse(null);
            if (idToTransfer != null) {
                Transferencia tr = juan.transferirTiquete(idToTransfer, maria, "pass123");
                System.out.println("[OK] Transferencia completada. ID transferencia: " + tr.getId());
            } else {
                System.out.println("[WARN] Juan no tiene tiquetes para transferir.");
            }
        } catch (UsuarioNoAutenticadoException | TiqueteNoTransferibleException e) {
            System.out.println("[ERROR] Transferencia fallida: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("[ERROR] Excepción en transferencia: " + e.getMessage());
        }

        System.out.println("\nTiquetes de Maria tras transferencia:");
        maria.imprimirTiquetes();

        System.out.println("\n== Historia 3: Intento de compra con saldo insuficiente ==");
        try {
            Transaccion t2 = maria.comprarTiquetes(concierto, vip, 2, vip.getPrecioBase());
            System.out.println("[OK] Compra completada: " + t2.getId());
        } catch (SaldoInsuficienteException e) {
            System.out.println("[ERROR] Compra fallida (saldo): " + e.getMessage());
        } catch (Exception e) {
            System.out.println("[ERROR] Compra fallida: " + e.getMessage());
        }

        System.out.println("\n== Historia 4: Generar reporte financiero (organizador) ==");
        ReporteFinanciero r = new ReporteFinanciero("mensual", "Resumen de ventas
ejemplo");
```

```
        org.generarReporte(r);
        System.out.println("[OK] Reporte generado: " + r.getIdReporte());

        System.out.println("\n== Prueba de restricción: administrador NO puede comprar ==");
        System.out.println("[INFO] Administrador no tiene método de compra (restricción de
negocio).");

        System.out.println("\n=== FIN DEMO ===");
    }
}
```

---

3) Cómo ejecutar en Eclipse (paso a paso)

1. `File -> New -> Java Project` -> nombre: **BoletaMaster**.
2. Dentro de `src`, crea paquetes: `logica`, `logica.eventos`, `logica.tickets`, `logica.finanzas`, `excepciones`, `presentacion`.
3. Crea los archivos Java en cada paquete (pega el contenido que te entregué).
4. Guarda todo.
5. Clic derecho sobre `presentacion/DemoMain.java` → `Run As` → **Java Application**.
6. Mira la pestaña **Console** (abajo) para ver la ejecución y los mensajes.

---

4) Qué imprime y cómo lo maneja

- Los errores de negocio (saldo insuficiente, límite de tiquetes, intento de transferencia inválida) se **lanzán como excepciones** y se capturan en `DemoMain` para **imprimir mensajes claros**; esto demuestra manejo de errores y permite continuar la demo sin detener el programa.
- Los mensajes están redactados en un estilo profesional / narrativo (fácil de leer para evaluadores).
- El flujo simula las **historias de usuario** solicitadas (compra, transferencia, crear evento, generar reporte).

---

5) Notas finales y siguientes pasos (opcionales)

- Si quieres, puedo:
    - Añadir persistencia simple (guardar objetos a JSON/CSV usando Gson).
    - Añadir tests JUnit que verifiquen reglas de negocio.
    - Convertir los paquetes `logica.*` a un `com.boletamaster.*` más formal (si el curso lo prefiere).
    - Generar un archivo `README.md` con instrucciones exactas y captura de consola ejemplo.

¿Quieres que **inyecte persistencia a archivos JSON** ahora (usando una única dependencia `gson`), o prefieres dejar la persistencia para la entrega 3?
Si quieres persistencia, te doy el código y el `pom.xml` o instrucciones para añadir `gson.jar`.