Comprensión de los Requerimientos

- Entender los requerimientos de un problema es una de las tareas más difíciles que enfrenta el ingeniero de software
- No parece tan difícil desarrollar un entendimiento claro de los requerimientos. Después de todo, ¿acaso no sabe el cliente lo que se necesita?
- Incluso si los clientes y los usuarios finales explican sus necesidades, éstas cambiarán mientras se desarrolla el proyecto

Definición de Requerimientos Funcionales

Definición

- Describen lo que el sistema debe hacer para cumplir con su propósito.
- Son especificaciones detalladas de funciones y características que el software debe incluir para satisfacer las necesidades del usuario o del negocio.

En principio, la especificación de los requerimientos funcionales de un sistema debe ser completa y consistente.

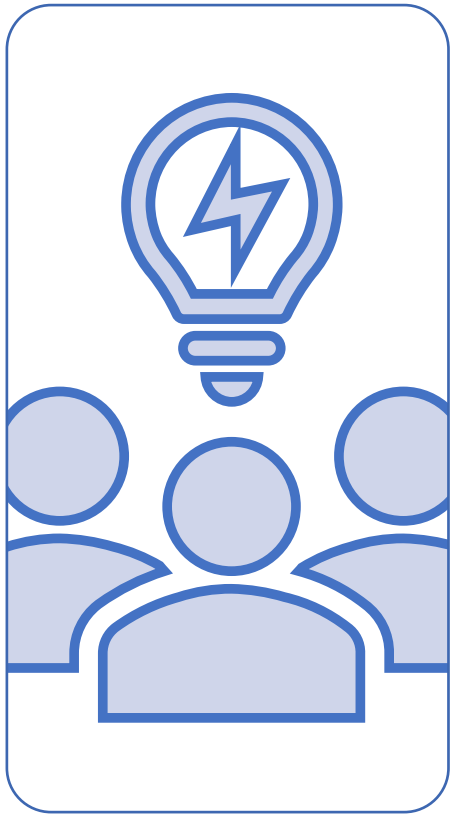
- **Totalidad** significa que deben definirse todos los servicios requeridos por el usuario.
- **Consistencia** quiere decir que los requerimientos tienen que evitar definiciones contradictorias.

Estos requerimientos pueden incluir:

- **Operaciones principales**: Definen las acciones básicas que el sistema debe ejecutar, como el registro de usuarios, procesamiento de datos o generación de informes.
- **Validaciones y restricciones**: Establecen reglas sobre cómo los datos deben ser ingresados, procesados y almacenados.
- **Interacciones con el usuario**: Especifican cómo el software debe responder a las acciones de los usuarios, incluyendo menús, botones y mensajes.
- **Integraciones con otros sistemas**: Detallan cómo debe comunicarse con otras aplicaciones o bases de datos.

En esencia, los requerimientos funcionales describen el "qué" del sistema.

Ejemplo de Requerimientos Funcionales



Sistema de venta de Música Online

- Los usuarios comprarán créditos para adquirir canciones.
- El sistema debe registrar la información de los usuarios y los créditos que poseen.
- Los usuarios buscarán las canciones que deseen y las pagarán con créditos.
- El sistema debe almacenar información sobre las canciones que se pueden adquirir y su precio en Créditos.
- El sistema deberá sugerir canciones acorde a los intereses del usuario.

Definición de Requerimientos No Funcionales

Definición

- Describen las características y restricciones del sistema que no están directamente relacionadas con sus funciones, sino con cómo debe operar.
- Surgen a través de necesidades del usuario, debido a restricciones presupuestales, políticas de la organización, necesidad de interoperabilidad con otro software, o factores externos como regulaciones de seguridad o legislación sobre privacidad.

Algunos ejemplos de requerimientos no funcionales:

- Rendimiento: Define la velocidad de respuesta del sistema, tiempo de procesamiento y uso de recursos.
- Tamaño: Establece límites de almacenamiento o capacidad de memoria RAM del servidor.
- Seguridad: Especifica cómo se protege la información y los accesos de los usuarios.
- Usabilidad: Se refiere a la facilidad con la que los usuarios pueden interactuar con el software.
- Fiabilidad: Se refiere a su capacidad para funcionar correctamente y sin fallos durante un período determinado y bajo condiciones específicas.
- Escalabilidad: Indica la capacidad del sistema para manejar un mayor volumen de usuarios o datos sin degradarse.
- Mantenimiento y actualizaciones: Establece la facilidad con la que se pueden realizar cambios y mejoras en el sistema.
- Compatibilidad: Define con qué plataformas, navegadores y dispositivos debe funcionar el software.

Estos requerimientos suelen ser tan importantes como los funcionales, ya que impactan directamente en la experiencia del usuario y la confiabilidad del sistema.

Introducción

¿Quién lo hace?

- Los ingenieros de software

¿Por qué es importante?

- Diseñar y construir un elegante programa de cómputo que resuelva el problema equivocado no satisface las necesidades de nadie.

¿Cuáles son los pasos?

- La ingeniería de requerimientos comienza con la concepción, indagación, elaboración, negociación y finalmente se especifica el problema.

¿Cuál es el producto final?

- El objetivo de los requerimientos de ingeniería es proporcionar a todas las partes un entendimiento escrito del problema.

¿Cómo me aseguro de que lo hice bien?

- Se revisan con los participantes los productos del trabajo de la ingeniería de requerimientos a fin de asegurar que lo que se aprendió es lo que ellos quieren decir en realidad.

Ingeniería de Requerimientos

El conjunto amplio de tareas y técnicas que llevan a entender los requerimientos se denomina ingeniería de requerimientos.

La ingeniería de requerimientos proporciona el mecanismo apropiado para entender:

- lo que desea el cliente,
- analizar las necesidades,
- evaluar la factibilidad,
- negociar una solución razonable,
- especificar la solución sin ambigüedades,
- validar la especificación y
- administrar los requerimientos a medida de que se transforman en un sistema funcional.

Ingeniería de Requerimientos …

La ingeniería de requerimientos incluye siete tareas:

Concepción.

- La mayor parte de proyectos comienzan cuando se identifica una necesidad del negocio o se descubre un nuevo mercado o servicio potencial.
- En esta tarea se establece el entendimiento básico del problema, las personas que quieren una solución, la naturaleza de la solución que se desea, así como la eficacia de la comunicación y colaboración preliminares entre los otros participantes y el equipo de software.

Indagación.

- Preguntar al cliente, a los usuarios cuáles son los objetivos para el sistema o producto, qué es lo que va a lograrse, cómo se ajusta el sistema o producto a las necesidades del negocio y cómo va a usarse el sistema o producto en las operaciones cotidianas.
- Surgen problemas de alcance (Límites mal definidos), entendimiento(No están seguros de lo que necesitan) y volatilidad (Cambios en el tiempo)

Ingeniería de Requerimientos …

Elaboración.

- Esta tarea se centra en desarrollar un modelo refinado de los requerimientos que identifique distintos aspectos de la función del software, su comportamiento e información.
- La elaboración está motivada por la creación y mejora de escenarios de usuario que describan cómo interactuará el usuario final (y otros actores) con el sistema.

Negociación.

- No es raro que los clientes y usuarios pidan más de lo que puede lograrse dado lo limitado de los recursos del negocio.
- También es relativamente común que distintos clientes o usuarios propongan requerimientos conflictivos con el argumento de que su versión es “esencial para nuestras necesidades especiales”.

Especificación.

- Una especificación puede ser un documento escrito, un conjunto de modelos gráficos, un modelo matemático formal, un conjunto de escenarios de uso, un prototipo o cualquier combinación de éstos.
- Se sugiere que para una especificación debe desarrollarse y utilizarse una “plantilla estándar” ya que esto conduce a requerimientos presentados en forma consistente y por ello más comprensible.

Ingeniería de Requerimientos …

Validación.

- La validación de los requerimientos analiza la especificación a fin de garantizar que todos ellos han sido enunciados sin ambigüedades; que se detectaron y corrigieron las inconsistencias, las omisiones y los errores, y que los productos del trabajo se presentan conforme a los estándares establecidos para el proceso, el proyecto y el producto.

Administración de los requerimientos.

- La administración de los requerimientos es el conjunto de actividades que ayudan al equipo del proyecto a identificar, controlar y dar seguimiento a los requerimientos y a sus cambios en cualquier momento del desarrollo del proyecto

Formato de especificación de requerimientos de software

Una especificación de requerimientos de software (ERS) es un documento que se crea cuando debe especificarse una descripción detallada de todos los aspectos del software que se va a elaborar.

1. Introducción

- 1.1 Propósito
- 1.2 Convenciones del documento
- 1.3 Alcance del proyecto
- 1.4 Referencias

2. Descripción general

- 2.1 Características del producto
- 2.2 Clases y características del usuario
- 2.3 Ambiente de operación
- 2.4 Restricciones de diseño e implementación
- 2.5 Suposiciones y dependencias

3. Características del sistema

- 3.1 Descripción de requerimientos funcionales

4. Requerimientos de la interfaz externa

- 4.1 Interfaces de usuario
- 4.2 Interfaces del hardware
- 4.3 Interfaces del software
- 4.4 Interfaces de las comunicaciones

5. Otros requerimientos no funcionales

- 5.1 Requerimientos de desempeño
- 5.2 Requerimientos de seguridad
- 5.3 Requerimientos de estabilidad
- 5.4 Atributos de calidad del software

Apéndice A: Glosario

Apéndice B: Modelos de análisis

Requerimientos del usuario y requerimientos del sistema

- Los requerimiento del usuario son muy generales, mientras que los requerimientos del sistema ofrecen información más específica sobre los servicios y las funciones del sistema.

Definición del requerimiento del usuario

Si se sabe que un paciente es alérgico a algún fármaco en particular, entonces la prescripción de dicho medicamento dará como resultado un mensaje de advertencia que se emitirá al usuario del sistema.
Si quien prescribe ignora una advertencia de alergia, deberá proporcionar una razón para ello.

Especificación de los requerimientos del sistema

1.1 El sistema debe registrar información de pacientes incluidas alergias
1.2 Prescribir medicamentos por separado si el paciente es alérgico a una de ellas.
1.3 El sistema deberá emitir automáticamente una advertencia por cada medicamento al que sea alérgico el paciente.
1.4 El sistema debe permitir prescribir medicamentos al que el paciente es alérgico, siempre y cuando se explique el motivo de tal hecho.

Desarrollo de casos de uso

Definición

- Es una técnica utilizada para capturar los requisitos de un sistema. Describe cómo un usuario o sistema externo interactúa con el software para lograr un objetivo específico.

Características

- Responde al comportamiento de un sistema desde la perspectiva del usuario para alcanzar los objetivos del negocio.
- Describe lo que el sistema hace, y no la manera en que lo hace.
- Un caso de uso narra una historia estilizada sobre cómo interactúa un usuario final (que tiene cierto número de roles posibles) con el sistema en circunstancias específicas.
- Es un texto narrativo, un lineamiento de tareas o interacciones, una descripción basada en un formato o una representación diagramática.
- Sin importar su forma, un caso de uso ilustra el software o sistema desde el punto de vista del usuario final.
- Con una definición más formal, un actor es cualquier cosa que se comunique con el sistema o producto y que sea externo a éste.

Desarrollo de casos de uso …

Una vez identificados los actores, es posible desarrollar casos de uso, los cuales deben responder a preguntas como:

- ¿Quién es el actor principal y quién(es) el(los) secundario(s)?
- ¿Cuáles son los objetivos de los actores?
- ¿Qué precondiciones deben existir antes de comenzar la historia?
- ¿Qué tareas o funciones principales son realizadas por el actor?
- ¿Qué excepciones deben considerarse al describir la historia?
- ¿Cuáles variaciones son posibles en la interacción del actor?
- ¿Qué información del sistema adquiere, produce o cambia el actor?
- ¿Tendrá que informar el actor al sistema acerca de cambios en el ambiente externo?
- ¿Qué información desea obtener el actor del sistema?
- ¿Quiere el actor ser informado sobre cambios inesperados?

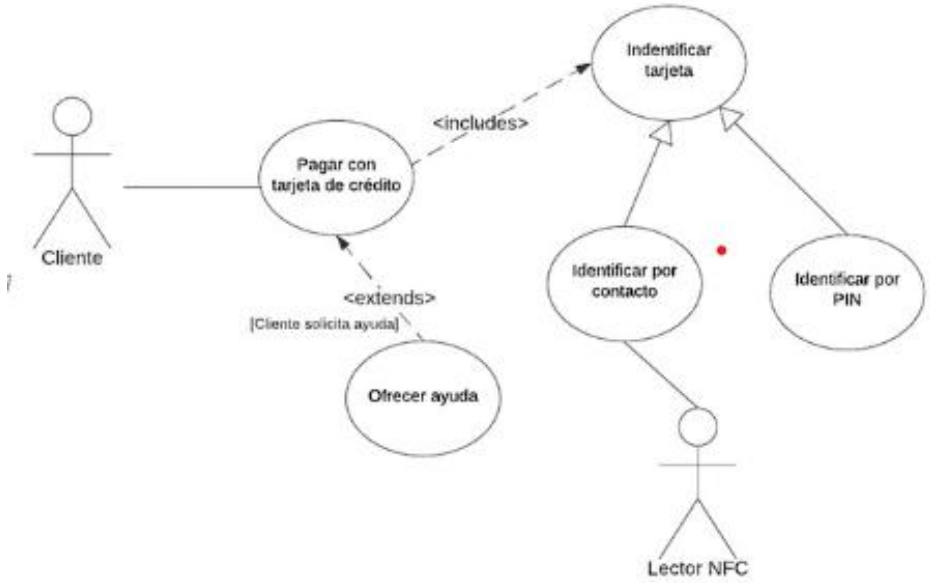
Desarrollo de casos de uso …

Asociaciones

- Hay una asociación entre un actor y un caso de uso si el actor interactúa con el sistema para llevar a cabo el caso de uso.

Tipos de Asociaciones

- <<include>> Un caso de uso base incorpora explícitamente a otro caso de uso en un lugar especifico en dicho caso base. Se suele utilizar para encapsular un comportamiento parcial común a varios casos de uso.
- <<exclude>> Se refiere a situaciones donde se evita la ejecución de un caso de uso bajo ciertas condiciones

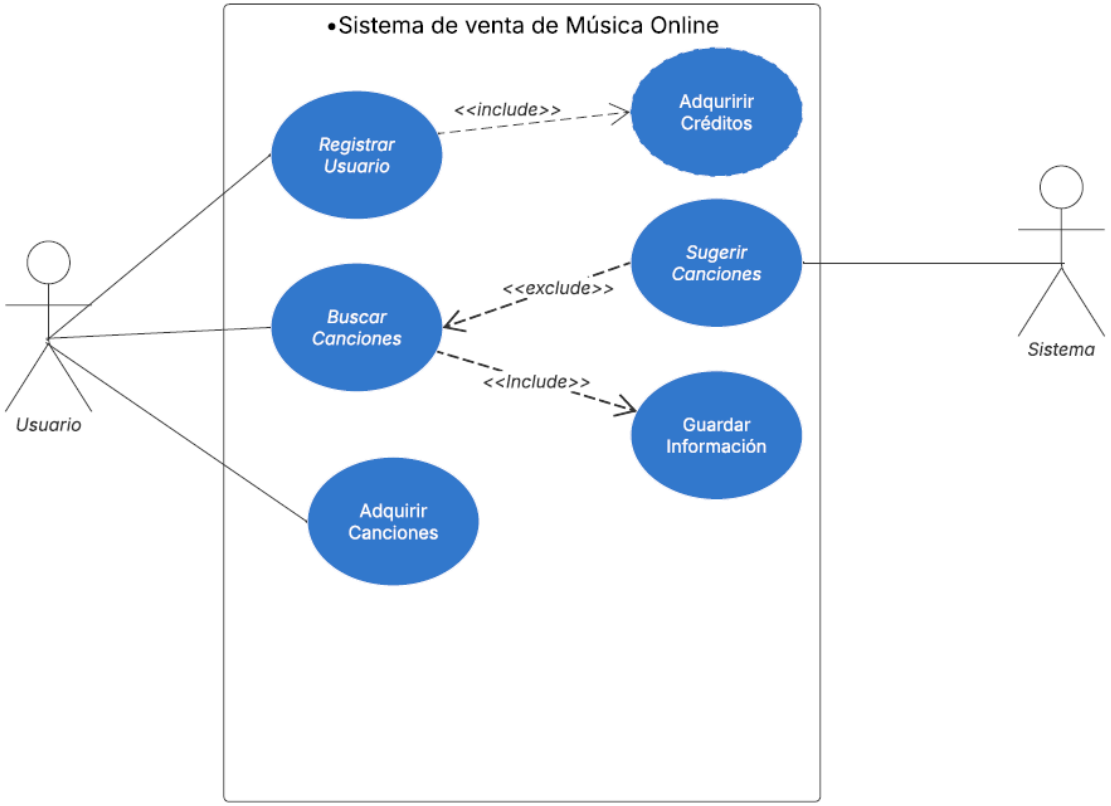


Código	Caso de Uso	Descripción RF

Ejemplo: Casos de uso …

Sistema de venta de Música Online

- Requerimientos funcionales para cada caso de uso



Código	Caso de Uso	Descripción RF
RF01	Registrar Usuario	El sistema debe registrar la información de los usuarios y los créditos que poseen.
RF02	Adquirir Créditos	Los usuarios comprarán créditos para adquirir canciones
RF03	Buscar Canciones	Los usuarios buscarán las canciones que deseen.
RF04	Sugerir Canciones	El sistema deberá sugerir canciones acorde a los intereses del usuario.
RF05	Guardar Información	El sistema debe almacenar información sobre las canciones que se pueden adquirir y su precio en Créditos.
RF06	Adquirir Canciones	Los usuarios adquirirán las canciones y las y las pagarán con créditos.

Tarea:

Ejercicio

- **Caso de uso**: Acceder a la vigilancia con cámaras por internet, mostrar vistas de cámaras (AVC-MVC)
- **Actor**: propietario
- Si estoy en una localidad alejada, puedo usar cualquier PC con un software de navegación apropiado para entrar al sitio web de Productos CasaSegura. Introduzco mi identificación de usuario y dos niveles de claves; una vez validadas, tengo acceso a toda la funcionalidad de mi sistema instalado. Para acceder a la vista de una cámara específica, selecciono “vigilancia” de los botones mostrados para las funciones principales. Luego selecciono “escoger una cámara” y aparece el plano de la casa. Después elijo la cámara que me interesa. Alternativamente, puedo ver la vista de todas las cámaras simultáneamente si selecciono “todas las cámaras”. Una vez que escojo una, selecciono “vista” y en la ventana que cubre la cámara aparece una vista con velocidad de un cuadro por segundo. Si quiero cambiar entre las cámaras, selecciono “escoger una cámara” y desaparece la vista original y de nuevo se muestra el plano de la casa. Después, selecciono la cámara que me interesa. Aparece una nueva ventana de vistas.

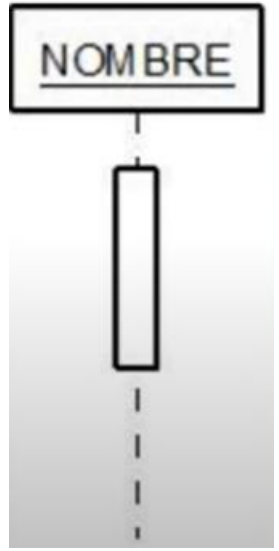
Desarrollar el requerimiento del sistema.

Desarrollar el Diagrama de Casos de uso

Diagramas de secuencia

- Diagramas de secuencia
  - Es un esquema conceptual que permite representar el comportamiento de un sistema
  - Emplea la especificación de los objetos que se encuentran en un escenario y la secuencia de mensajes intercambiados entre ellos, con el fin de llevar a cago una transacción del sistema.

- Los objetos se colocan cerca de la parte superior del diagrama de izquierda a derecha.



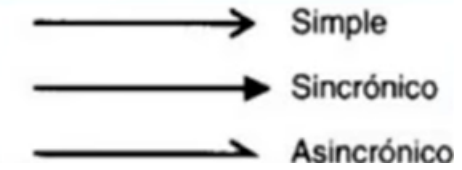
Diagramas de secuencia

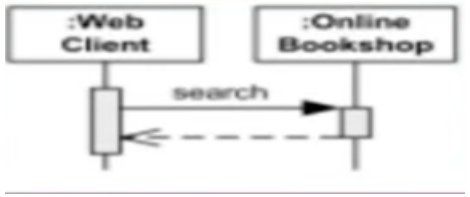
- Línea de vida
  - Representa un participante individual en un diagrama de secuencia.
  - Usualmente tiene un rectángulo.

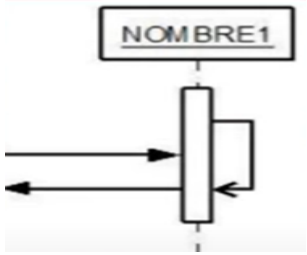
- Mensaje
  - Un mensaje que va de un objeto a otro pasa de la línea de vida de un objeto a otro.
  - Un objeto puede enviarse un objeto a si mismo, es decir de su línea de vida a su propia línea de vida

Diagramas de secuencia

- Tipos de mensajes
  - Mensaje simple: Es la transferencia del control de un objeto a otro.
  - Mensaje síncrono: Es cuando el objeto espera la respuesta a ese mensaje antes de continuar con su trabajo.
  - Mensaje asíncrono: es cuando el objeto no espera la respuesta a ese mensaje antes de continuar.
- Recursividad: Es una operación que se invoca a si misma







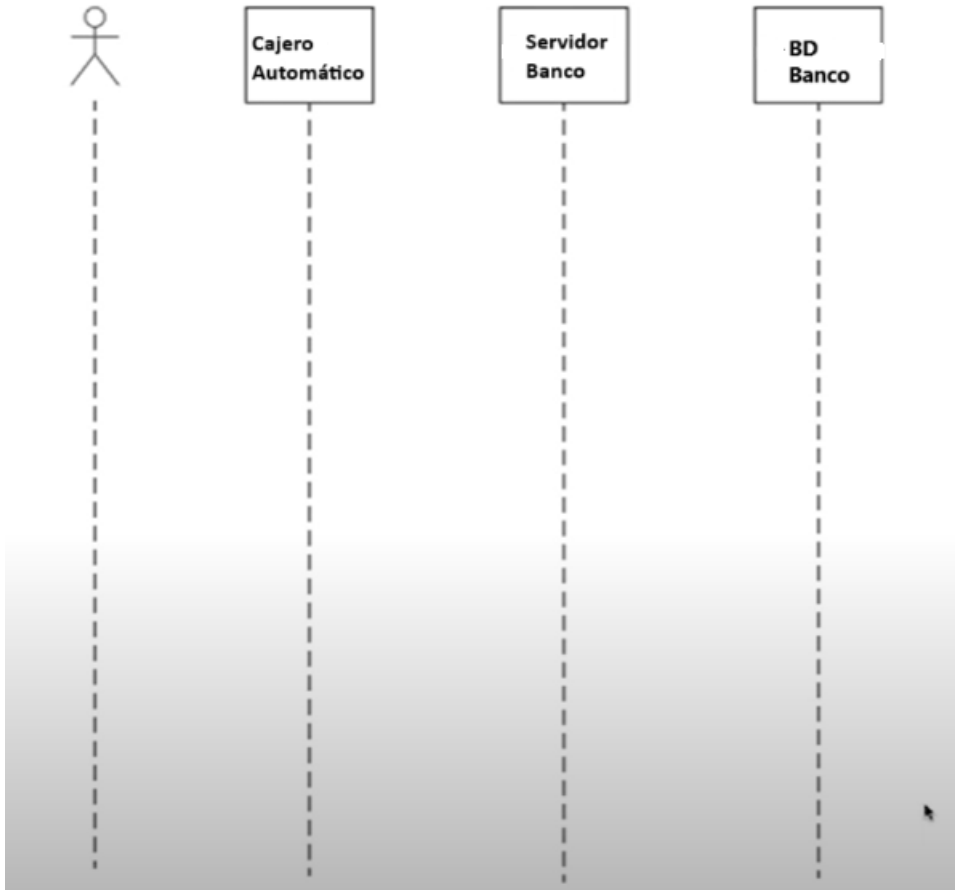
Ejemplo

Interacciones para retirar dinero de un cajero automático

Primero dibujamos las partes del diagrama:

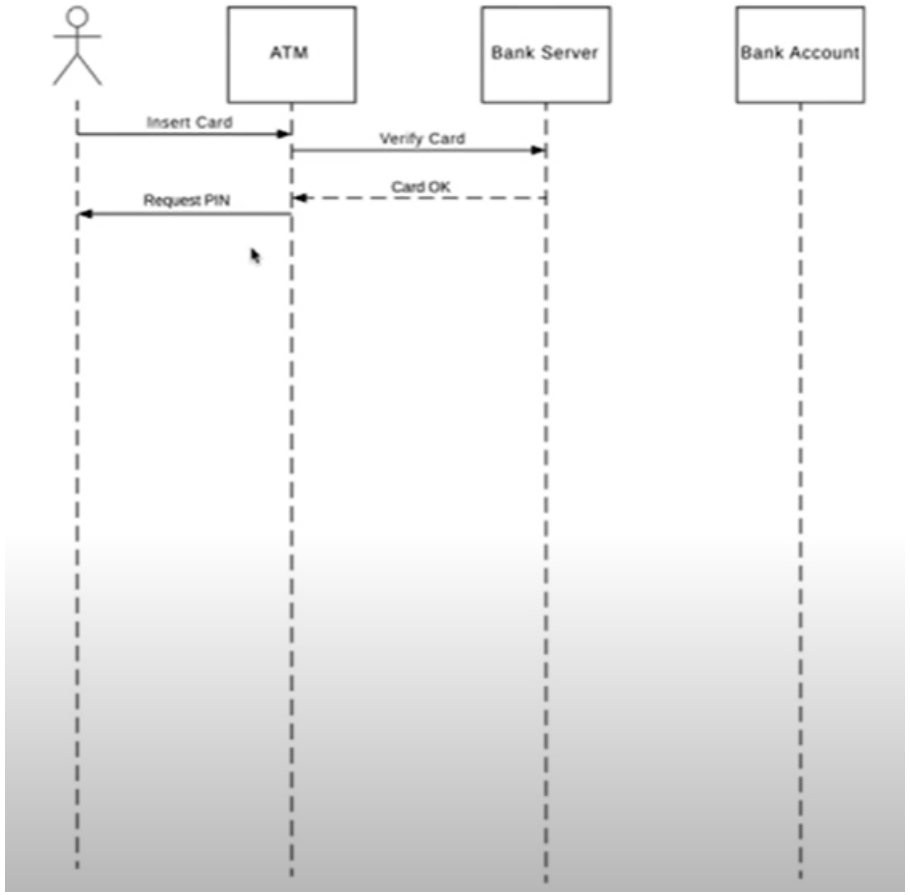
- Si es un sistema, entonces todas las partes de éste.
- Si es código, entones sería todas las clases que forman el programa.

Segundo dibujamos las líneas de vida

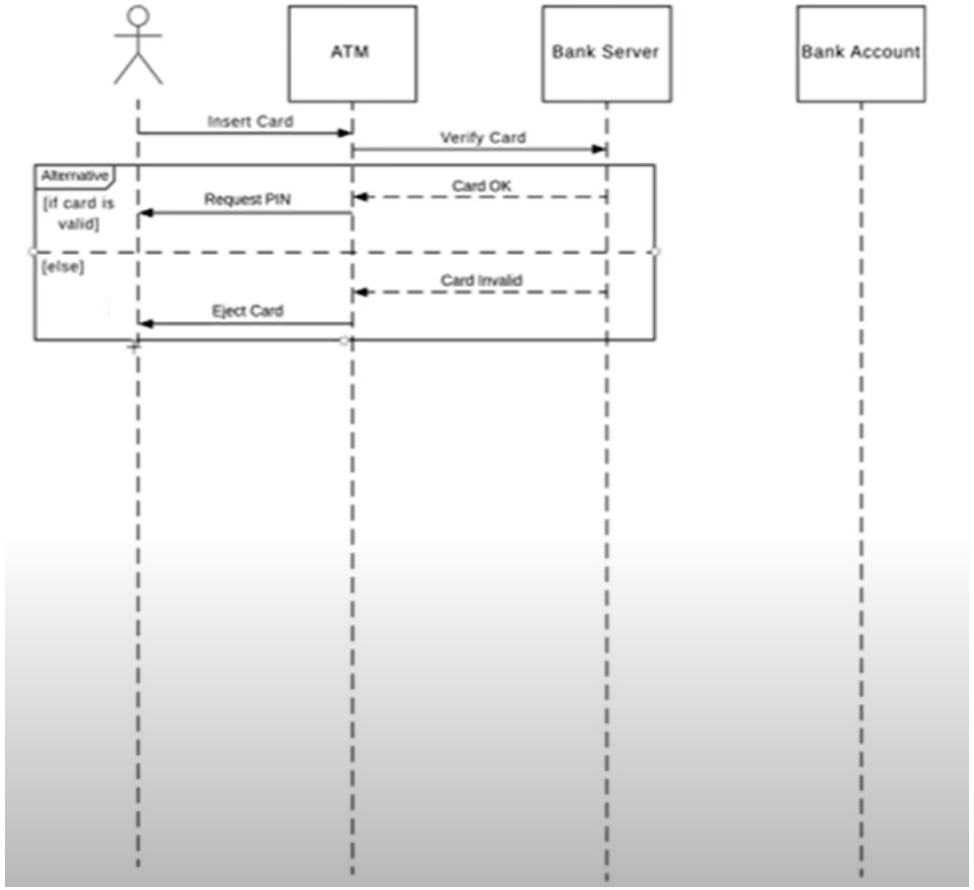


Ejemplo …

- Dibujamos los mensajes

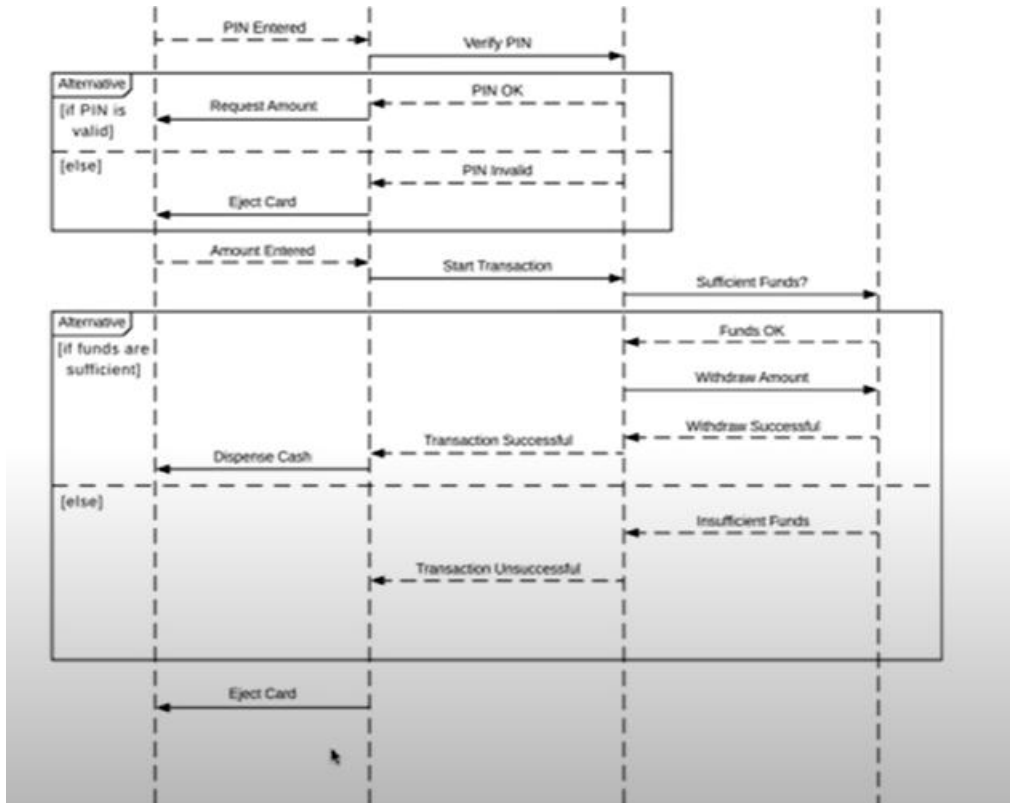


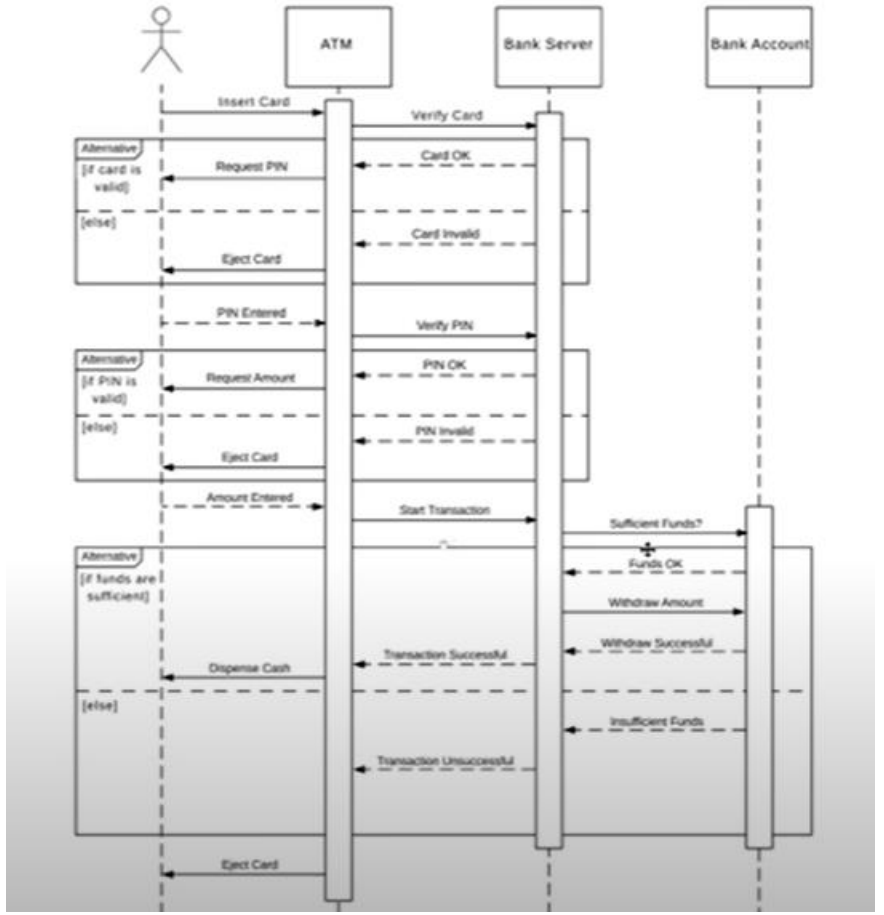
- Que pasa cuando la tarjeta es inválida?
- Usamos un cuadro alternativo
- Tarjeta es válida o Tarjeta es inválida



Ejemplo …

- Activamos las cajas





Clases

Definición

- En programación orientada a objetos (POO), una clase es una plantilla o modelo a partir del cual se crean objetos. Define las propiedades (atributos) y comportamientos (métodos) que tendrán los objetos de esa clase.
- La clase define las reglas; los objetos expresan los hechos.
- La clase define que puede ser; el objeto describe que es.

Características clave de una clase:

- **Atributos**: Representan las características del objeto (ej. "nombre", "edad" en una clase "Persona").
- **Métodos**: Son las acciones que el objeto puede realizar (ej. "caminar", "hablar").
- **Encapsulación**: Controla el acceso a los atributos y métodos mediante modificadores de acceso como public, private, y protected.
- **Herencia**: Permite que una clase hija herede atributos y métodos de una clase padre.
- **Polimorfismo**: Permite que métodos en distintas clases tengan el mismo nombre pero diferentes implementaciones.

Clases …

La encapsulación:

- Es un principio de la programación orientada a objetos que restringe el acceso directo a los datos de una clase, permitiendo que solo métodos específicos puedan modificar o acceder a ellos.

```
public class Main {
    public static void main(String[] args) {

        // Accediendo a los atributos mediante métodos
        System.out.println("Nombre: " + persona1.getNombre());
        System.out.println("Edad: " + persona1.getEdad());

        // Modificando los atributos con métodos setter
        persona1.setNombre("Peter ");
        persona1.setEdad(35);
    }
}
```

```
public class Persona {
    // Atributos privados (no accesibles directamente)
    private String nombre;
    private int edad;

    // Métodos getter para acceder a los atributos
    public String getNombre() {
        return nombre;
    }
    public int getEdad() {
        return edad;
    }

    // Métodos setter para modificar los atributos
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public void setEdad(int edad) {
        if (edad > 0) {  // Validación simple
            this.edad = edad;
        }
    }
}
```

Clases …

La herencia:

- Es un principio fundamental de la programación orientada a objetos que permite que una subclase herede atributos y métodos de una superclase.

```
// Clase padre (superclase)
class Animal {
    String nombre;

    // Método común
    public void hacerSonido() {
        System.out.println("El animal hace un
sonido.");
    }
}
```

```
// Clase hija (subclase) que hereda de Animal
class Perro extends Animal {

    // Método propio de la subclase
    public void ladrar() {
        System.out.println(nombre + " está ladrando.");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Perro miPerro = new Perro("Firulais");

        miPerro.hacerSonido(); // Heredado de Animal
        miPerro.ladrar();      // Propio de Perro
    }
}
```

Clases …

La herencia:

- Es un principio fundamental de la programación orientada a objetos que permite que una subclase herede atributos y métodos de una superclase.

```
// Clase padre (superclase)
class Animal {
    String nombre;

    // Método común
    public void hacerSonido() {
        System.out.println("El animal hace un
sonido.");
    }
}
```

```
// Clase hija (subclase) que hereda de Animal
class Perro extends Animal {

    // Método propio de la subclase
    public void ladrar() {
        System.out.println(nombre + " está ladrando.");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Perro miPerro = new Perro("Firulais");

        miPerro.hacerSonido(); // Heredado de Animal
        miPerro.ladrar();      // Propio de Perro
    }
}
```

Clases …

El polimorfismo:

- Permite que una clase pueda utilizar métodos con el mismo nombre pero con diferentes implementaciones.

Hay dos tipos principales de polimorfismo:

- Polimorfismo en tiempo de compilación (sobrecarga de métodos)
- Polimorfismo en tiempo de ejecución (sobreescritura de métodos)

```
// Clase padre
class Animal {
    public void hacerSonido() {
        System.out.println("El animal hace un sonido.");
    }
}
// Clases hijas que sobrescriben el método hacerSonido()
class Perro extends Animal {
    @Override
    public void hacerSonido() {
        System.out.println("El perro ladra: ¡Guau guau!");
    }
}
```

```
class Gato extends Animal {
    @Override
    public void hacerSonido() {
        System.out.println("El gato maúlla: ¡Miau miau!");
    }
}
```

```
// Clase principal
public class Main {
    public static void main(String[] args) {
        Animal miAnimal = new Animal();
        Animal miPerro = new Perro();  // Polimorfismo
        Animal miGato = new Gato();    // Polimorfismo

        miAnimal.hacerSonido();  // Llamado al método de
Animal
        miPerro.hacerSonido();   // Llamado al método
sobrescrito en Perro
        miGato.hacerSonido();    // Llamado al método
sobrescrito en Gato
    }
}
```

Diagramas de Clases …

Definición

- Son una representación visual utilizada en la programación orientada a objetos para describir la estructura de un sistema. Son esenciales en el diseño de software, ya que permiten visualizar las clases, sus atributos, métodos y las relaciones entre ellas.

Conceptos principales:

- Clases: Representan entidades del sistema y contienen atributos (datos) y métodos (funcionalidades).
- Objetos: Instancias de una clase.
- Atributos: Características de una clase, como "color" en una clase "Vehículo".
- Métodos: Acciones que una clase puede realizar, como "acelerar" en "Automóvil".

Relaciones: Conexión entre clases.

- Asociación: Vincula clases sin una relación jerárquica estricta.
- Herencia: Una clase hija adquiere atributos y métodos de una clase padre.
- Agregación: Una clase contiene otra, pero sin dependencia total.
- Composición: Una clase depende completamente de otra.

Diagramas de Clases …

La Asociación

- Representa la relación entre dos clases. No implica dependencia estricta, lo que significa que ambas clases pueden existir de manera independiente.

```
// Clase Persona
class Persona {
    private String nombre;

    public Persona(String nombre) {
        this.nombre = nombre;
    }
    public String getNombre() {
        return nombre;
    }
}
// Clase Dirección (asociada con Persona)
class Direccion {
    private String ciudad;
    private String pais;
    private Persona propietario; // Relación de asociación
    public Direccion(String ciudad, String país, Persona propietario) {
        this.ciudad = ciudad;
        this.pais = pais;
        this.propietario = propietario;
    }
    public void mostrarDireccion() {
        System.out.println("Dirección: " + ciudad + ", " + pais);
        System.out.println("Propietario: " + propietario.getNombre());
    }
}

// Clase principal
public class Main {
    public static void main(String[] args) {
        Persona persona1 = new Persona("César");  // Creación de objeto
Persona
        Direccion direccion1 = new Direccion("Quito", "Ecuador",
persona1); // Asociación

        direccion1.mostrarDireccion(); // Accediendo a los datos
    }
}
```

Diagramas de Clases …

La agregación

- Es una relación entre clases donde una contiene otra, pero la clase contenida puede existir de manera independiente.
- Es una relación **menos fuerte** que la composición porque el objeto agregado no depende completamente de la clase principal.

```
// Clase independiente: Departamento
class Departamento {
    private String nombre;

    public Departamento(String nombre) {
        this.nombre = nombre;
    }
    public String getNombre() {
        return nombre;
    }
}
// Clase Empresa que contiene Departamento mediante
agregación
class Empresa {
    private String nombre;
    private Departamento departamento; // Relación de agregación
    public Empresa(String nombre, Departamento
departamento) {
        this.nombre = nombre;
        this.departamento = departamento; // Se pasa el objeto
como parámetro, no se crea dentro
    }
    public void mostrarInfo() {
        System.out.println("Empresa: " + nombre);
        System.out.println("Departamento: " +
departamento.getNombre());
    }
}
// Clase principal
public class Main {
    public static void main(String[] args) {
        // Se crea un objeto independiente de Departamento
        Departamento depto = new Departamento("Recursos
Humanos");
        // Se asocia con una Empresa
        Empresa empresa = new Empresa("TechCorp", depto);
        empresa.mostrarInfo();
    }
}
```

Diagramas de Clases …

La **composición**

- Es un tipo de relación en la que una clase contiene otra como parte esencial de su existencia.
- Si el objeto principal se elimina, los objetos secundarios también deben eliminarse.
- Esta relación es más fuerte que la **asociación** y la **agregación**.

```
// Clase Motor (Parte esencial de un Auto)
class Motor {
    private String tipo;
    public Motor(String tipo) {
        this.tipo = tipo;
    }
    public String getTipo() {
        return tipo;
    }
}
```

```
// Clase Auto (Contiene un Motor mediante composición)
class Auto {
    private String marca;
    private Motor motor; // Relación de composición: Auto *posee* un
Motor
    public Auto(String marca, String tipoMotor) {
        this.marca = marca;
        this.motor = new Motor(tipoMotor); // Creación del objeto Motor
dentro de Auto
    }
    public void mostrarInfo() {
        System.out.println("Auto: " + marca);
        System.out.println("Motor: " + motor.getTipo());
    }
}
// Clase principal
public class Main {
    public static void main(String[] args) {
        Auto miAuto = new Auto("Toyota", "V8"); // Creación del Auto y su
Motor
        miAuto.mostrarInfo();
    }
}
```

Diagramas de Clases …

Interfaz

- Es una estructura que define un conjunto de métodos sin implementar.
- Es un mecanismo que permite la abstracción y la implementación de **polimorfismo.**
- Solo contienen operaciones públicas

```
// Definición de una interfaz
interface Animal {
   void hacerSonido(); // Método abstracto (sin
implementación)
}
// Clase Perro que implementa la interfaz Animal
class Perro implements Animal {
   @Override
   public void hacerSonido() {
      System.out.println("El perro ladra: ¡Guau guau!");
   }
}
```

```
// Clase Gato que también implementa la interfaz Animal
class Gato implements Animal {
   @Override
   public void hacerSonido() {
      System.out.println("El gato maúlla: ¡Miau miau!");
   }
}
// Clase principal con el método main
public class Main {
   public static void main(String[] args) {
      Animal miPerro = new Perro();
      Animal miGato = new Gato();
      miPerro.hacerSonido();
      miGato.hacerSonido();
   }
}
```