Diseño de Sistemas de Software

¿Que es un sistema?

- Una unidad formada de diversos componentes o partes que dependen entre si para proveer una funcionalidad.
- Una totalidad formada de componentes coordinados para proveer funcionalidad.

¿Que comprende la creación de un sistema de software moderno?

- Adquirir requerimientos
- Análisis y diseño con orientación a objetos
- Patrones de software
- Control de versiones
- Sistemas de construcción de software
  - make
  - ant
  - maven
- Sistema de pruebas
  - JUnit

- Requerimientos:
  - Que queremos que el sistema haga
- Análisis y diseño:
  - Análisis ocurre en el espacio del problema
  - Diseño ocurre en el espacio de la solución
  - Definir como se le va a dar la funcionalidad al sistema
  - Que componentes de hardware y software necesitamos
  - Que hay que programar
- Patrones de software – ¿Que son?:
  - Fragmentos de diseño que resuelven problemas comunes

- Reglas y buenas practicas orientadas a objetos
- Uso de la experiencia de la comunidad de desarrolladores de software

Ejemplo:

Nombre del Patrón:	Experto Informacional
Problema:	Cual es el principio básico para asignar responsabilidades a objetos?
Solución:	Asigna la responsabilidad a la clase que tiene la información necesaria para cumplir con la responsabilidad.

- Control de Versiones:
  - Como organizar y controlar cada componente de software y el sistema completo en un grupo de desarrolladores mientras cada componente cambia por desarrollo o mantenimiento

- Construcción del código fuente
  - Transformar el código fuente a binario
  - Enlazar todos los fragmentos del sistema

- Pruebas del sistema
  - Como probar cada componente del sistema (pruebas unitarias, unit tests)
  - Como probar el sistema en su totalidad (pruebas de aceptación; pruebas para comprobar que el sistema implementa los requerimientos)
  - Como dividir óptimamente el presupuesto de pruebas del sistema (donde invertir mas tiempo)
  - ¿DDT (Design-Driven Testing) o TDD (Test-Driven Development)?
  - Pruebas de caja negra (black box testing)
  - Pruebas de caja blanca (white box testing)

Desarrollo de Software Ágil

- Individuos e interacciones por encima de procesos y herramientas
- Software funcionando versus documentación extensa
- Colaboración con el cliente versus negociaciones llena de jerga legal
- Respuesta a cambios versus el seguimiento estricto de un plan

Análisis y Diseño Orientado a Objetos

- Análisis enfatiza las características del problema en el campo del dominio
  - Uso del sistema por profesionales del campo
  - Especificaciones de los requerimientos
  - Funciones del sistema
- Diseño enfatiza una solución conceptual
  - Un modelo mental
  - Una generalización
  - Una abstracción
  - Satisface los requerimientos e ignora la implementación

En Particular: Análisis Orientado a Objetos

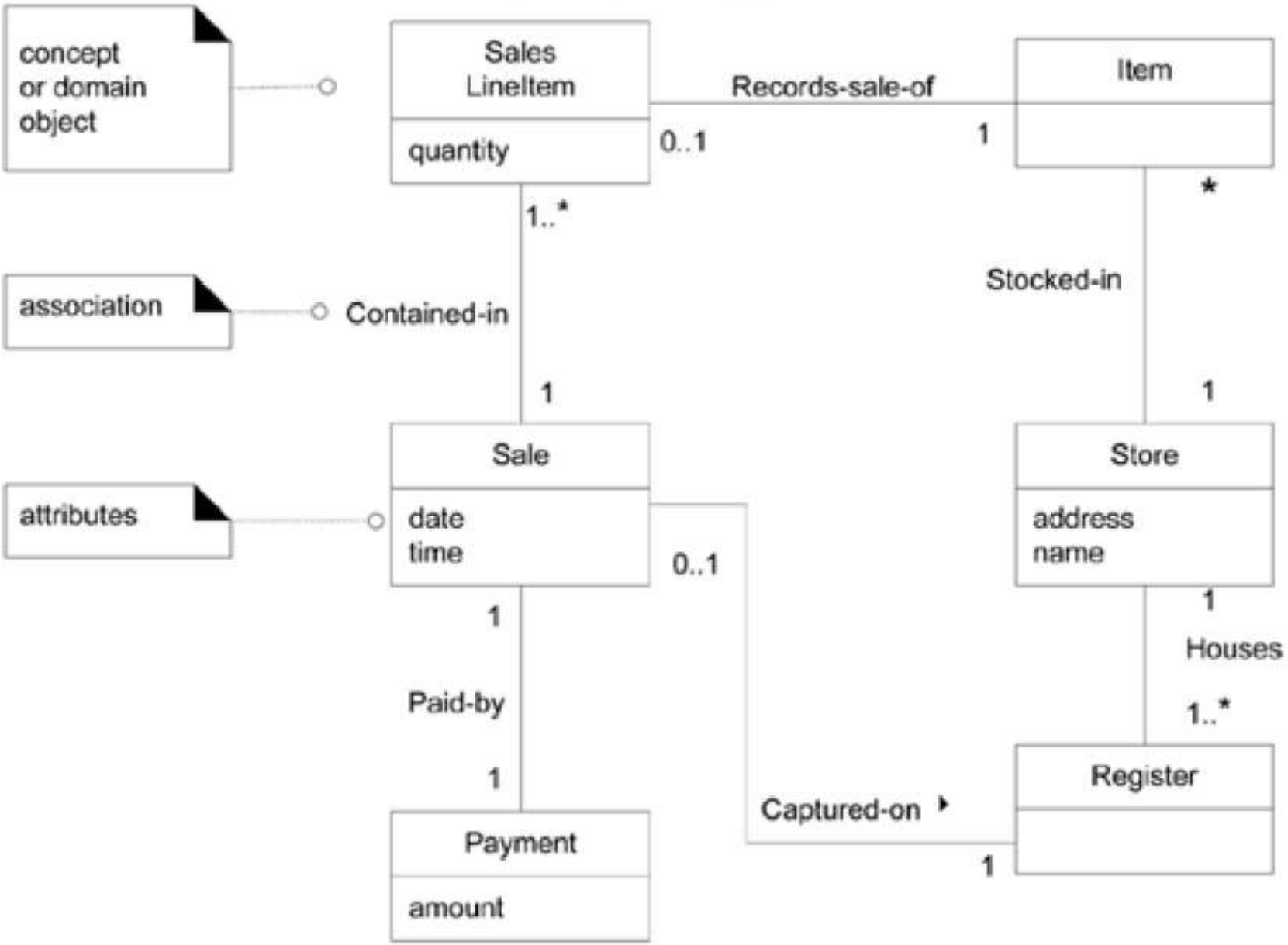
- Encontrar y descubrir objetos y conceptos en el dominio del problema
- Cada objeto tiene que tener cohesion; el grado que los elementos del objecto pertenecen juntos.
- Entre objetos tiene que haber bajo acoplamiento; o sea baja interdependencia entre objectos
- Un dominio es un campo de estudio con requerimientos comunes
- Ejemplo de un sistema informático que provee información de vuelos:
  - Avión
  - Vuelo
  - Piloto Automatico

Objecto de Dominio

- Conceptos importantes del campo donde vamos a proveer una solucion
- Una especie de glosario de terminos
- Son conceptos del mundo real y no de software
- Inspiran clases de software

1-26-2023

Por Ejemplo un Punto de Venta



¿Que es un Objeto?

- Una entidad que tiene estado, comportamiento e identidad.
  - **Estado** se refiere a las propiedades y el valor de las mismas a través del ciclo de vida del objeto.
  - **Comportamiento** se refiere a como el objeto actúa y reacciona en términos de cambios de estado y e invocación de sus métodos.
  - **Identidad** se refiere a la propiedad de un objeto que lo distingue de los demás objetos.

Características de Objetos

- Propiedades estáticas y dinámicas
  - por ejemplo el objeto **archivo** tiene las propiedades estáticas:
    - espacio
    - nombre
    - Contenido
  - los valores da cada una de las propiedades estáticas a través de la vida del objeto son las partes dinámicas del objeto

Responsabilidades del Objeto

- Los servicios que provee el objeto a los otros objetos.
- El orden en el cual otros objetos invocan las operaciones del objeto. El protocolo del objeto.
- El papel o rol que puede tener en el sistema:
  - Cliente: solo opera otros objetos
  - Servidor: solo puede ser operado por otros objetos
  - Proxy: puede ser operado o operar por otros objetos

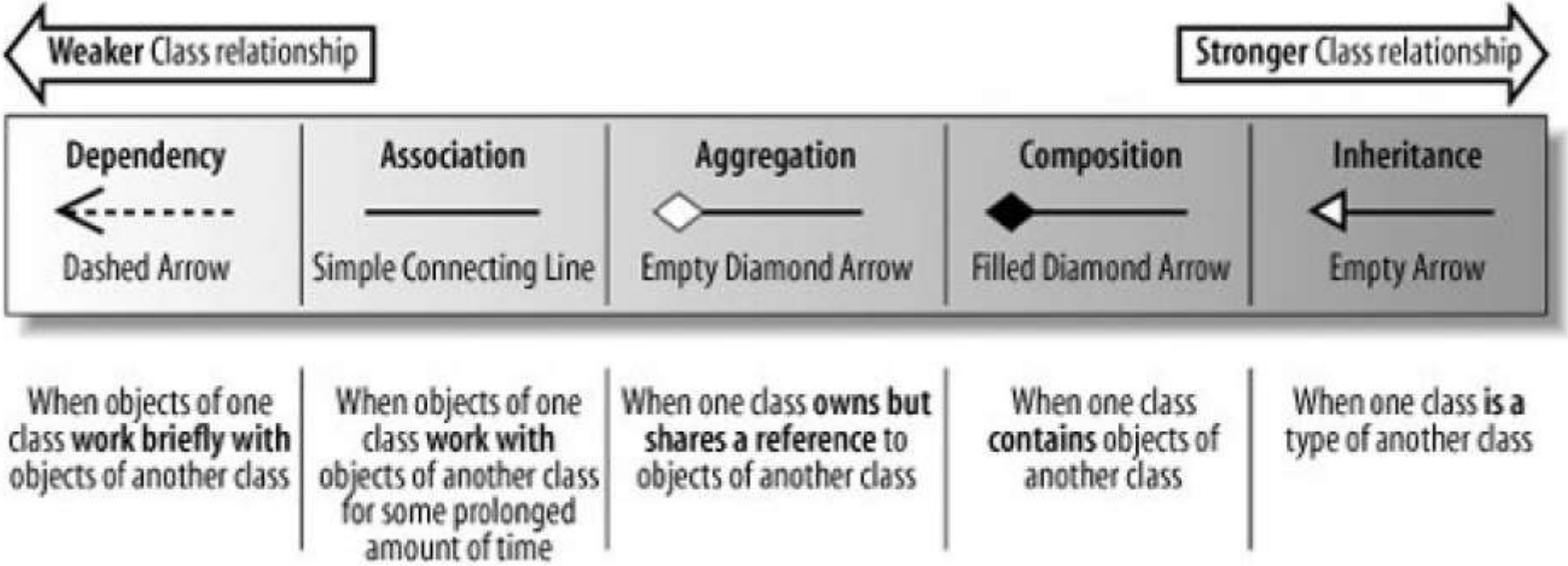
Interface e Implementación

- Una clase sirve como contrato entre una abstracción y sus clientes
- El contrato es captado por el interfaz de la clase.
  - Visión externa de la clase
  - Visión interna de la clase
- División del interfaz de la clase:
  - Public: una declaración accesible a todos los clientes.
  - Protected: una declaración accesible a la clase misma y sus subclases.
  - Private: una declaración accesible solo a la clase misma.
  - Package: una declaración accesible a todas las clases del paquete.

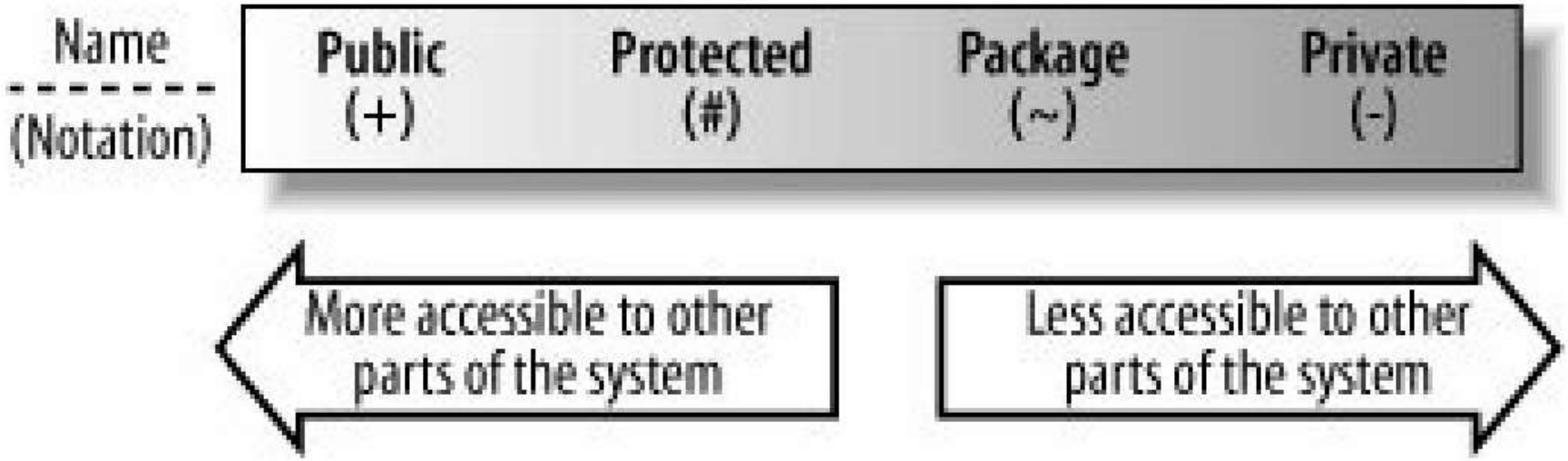
Relaciones Entre Objetos

- Enlaces: una relación entre objetos.
- Agregación: denota una jerarquía todo/parte, con la habilidad de navegar del todo a la parte.
- Cuando el tipo de agregación del todo/parte implica dependencia del ciclo de vida de la parte con el todo se llama composición; libro/capitulo, casa/cuarto, edificio/apt.

Tipos de Relaciones Entre Clases

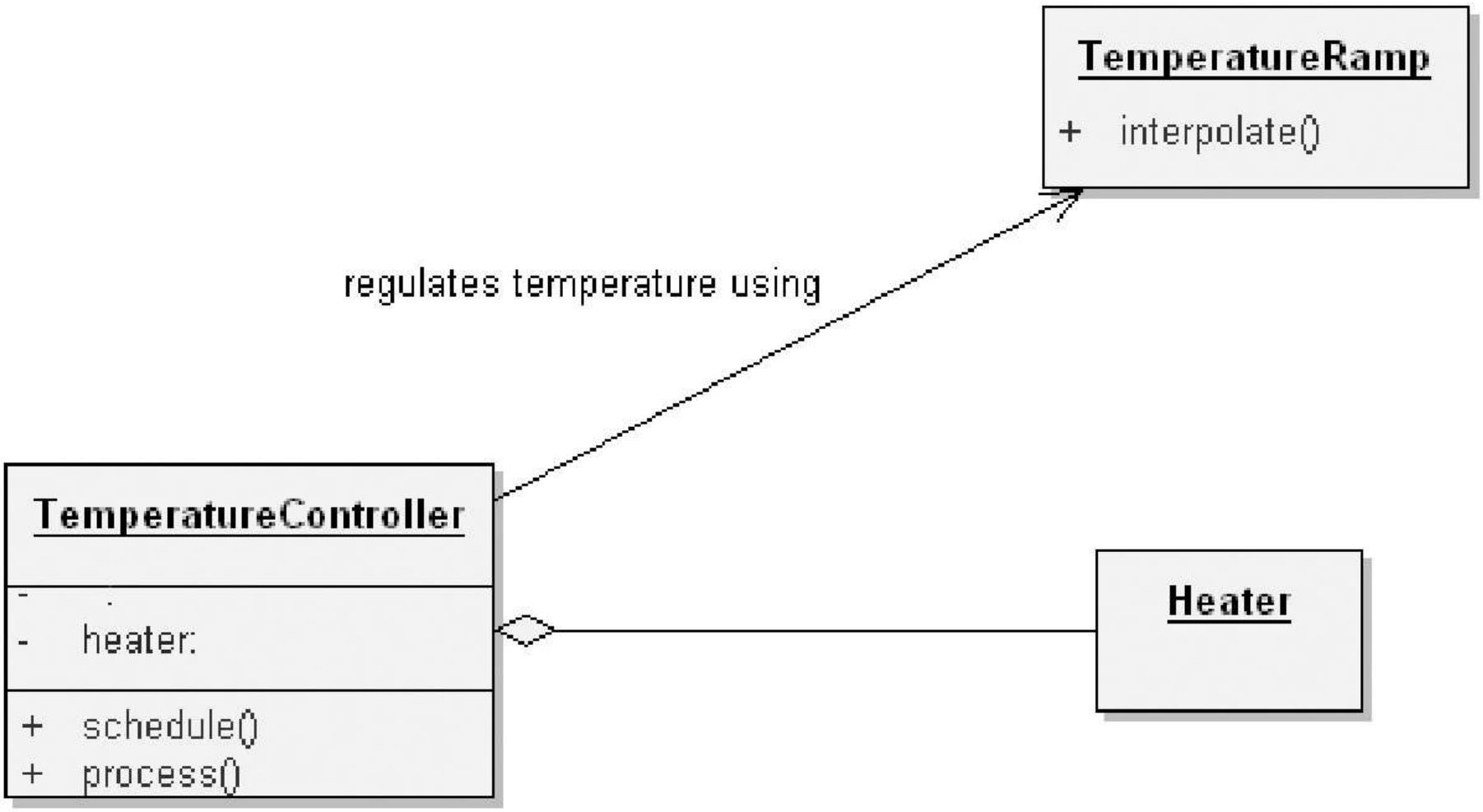


Visibilidad de los Componentes de Clases



Ejemplo

- Proceso industrial que requiere calentar y enfriar materiales en función de tiempo del proceso.
- TemperatureRamp: clase que mapea tiempo a temperatura del material.
- Heater: clase que controla el dispositivo que hace realidad el perfil de calentamiento y enfriamiento del material.
- TemperatureController: clase que sabe que perfil de calentamiento y enfriamiento hay que aplicar a algún material. Sabe cuando se ha enfriado el "heater" después de algún uso anterior. Tiempo de enfriamiento.


TemperatureRamp
+ interpolate()
regulates temperature using
TemperatureController
- heater:
+ schedule()
+ process()
Heater

Crear un Juego de Dados – Ejemplo con Diagramas Claves

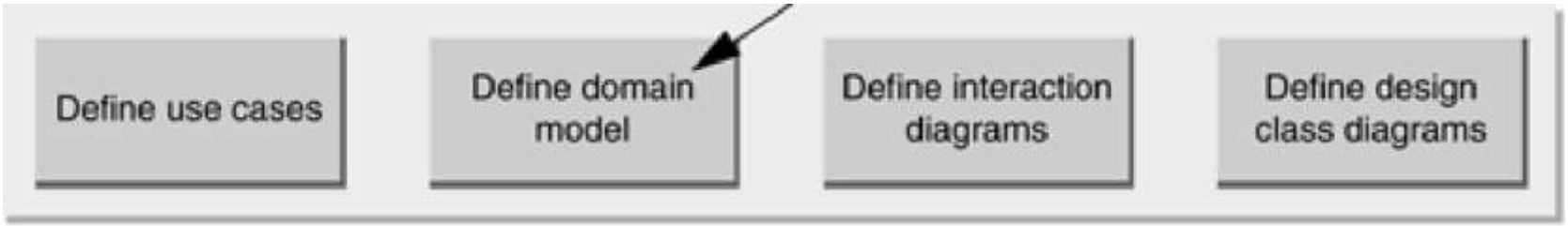
- Requerimientos descritos tradicionalmente
  - El juego de dados simula un jugador lanzando dos dados
  - Si el puntaje es siete el jugador gana
  - Si el puntaje es un numero que no es 7 el jugador pierde

Requerimientos Usando Casos de Uso

Jugar una partida de dados: Un jugador recoge y lanza los dados. Si el valor de las caras de los dados suman siete, gana; en otro caso, pierde.

Modelo del Dominio

- El análisis orientado a objetos crea una descripción del dominio desde la perspectiva de clasificación de objetos
- El dominio se descompone, identificando conceptos, atributos y asociaciones significativas.
- El producto de esta descomposición se puede expresar en un modelo de dominio.
- No son objetos de software, si no conceptos del mundo real.



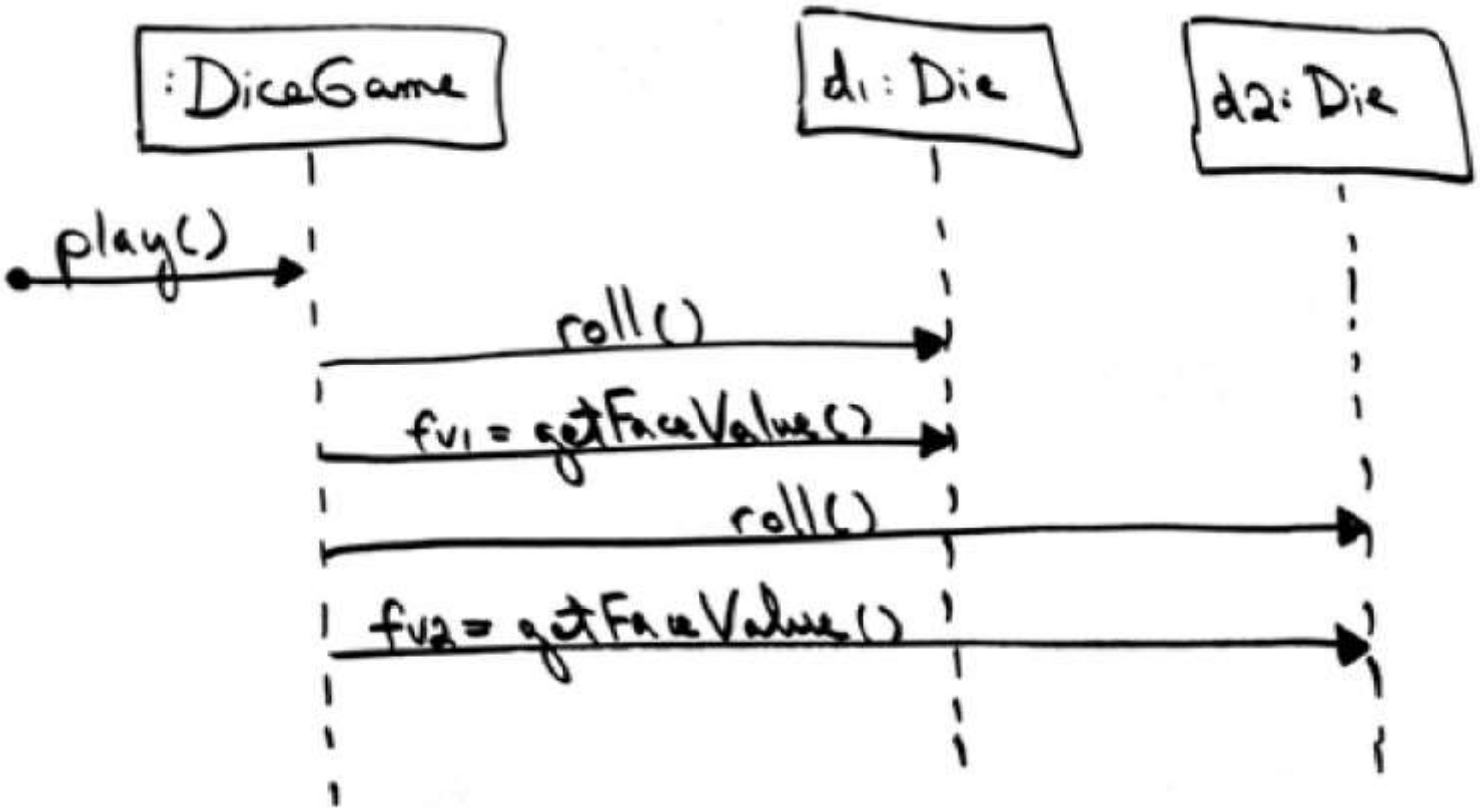
Modelo del Dominio de Juego de Dados

El modelo muestra los conceptos importantes Player, Die y DiceGame con sus asociaciones y atributos.

Diagramas de Interacción

- Muestra el flujo de mensajes de los objetos de software; invocación de métodos.
- Los diseños de los objetos de software, se inspiran en el dominio del mundo real.

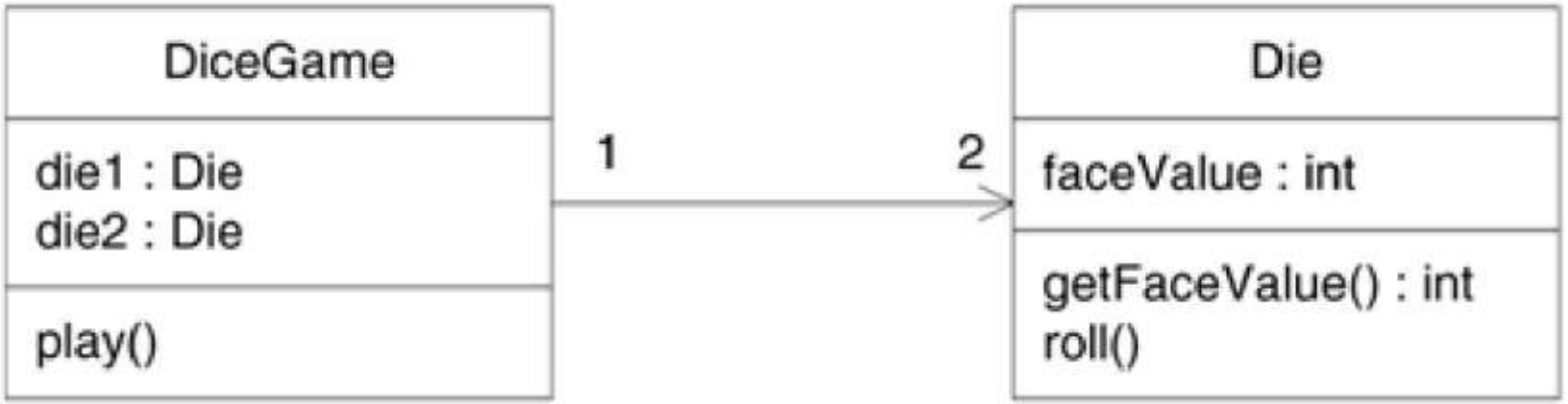
Diagrama de Interacción Parcial



Clases de Diseño

- Es una vista estática de las definiciones de las clases mediante un diagrama de clases de diseño.
- El diagrama de interacción conduce al diagrama de clases de diseño.
- El objeto DiceGame usa el método roll() del objeto Die
- El objeto Die requiere tener el método roll()

Diagrama de Clases



Términos

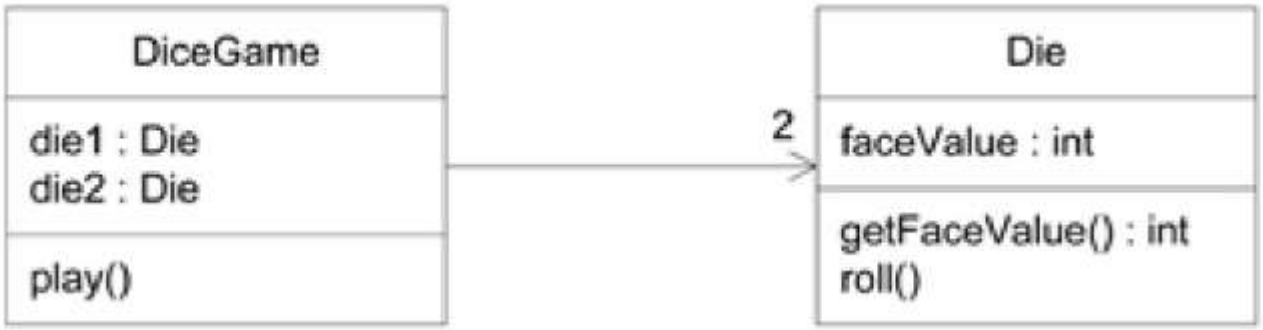
- Concepto: Una idea general. Algo concebido en la mente. Una idea abstracta.
- Abstracto: Expresar una propiedad, cualidad, atributo o relación vista aparte de las demás características de un objeto. Lo contrario de lo concreto.
- Abstracción: La simplificación de un objeto o concepto, manteniendo atributos relevantes al problema en mente.
- Artefacto: Algo tangible, producido como producto del proceso de desarrollo del sistema. Por ejemplo: diagramas de UML, planes del proyecto, lista de errores, etc.

- UML (Unified Modeling Language): el lenguaje unificado de modelado es un lenguaje gráfico para especificar, construir y documentar los artefactos de los sistemas de software, así como para el modelado del negocio y otros sistemas no software.
- Requerimientos Funcionales: describe lo que tiene que hacer el sistema.
- Requerimientos No Funcionales: describe como tiene que proveer ciertas características. Por ejemplo tiempos de respuesta, carga del sistema, tolerancia a errores, seguridad del sistema, usabilidad del sistema, estética del interfaz grafico, confiabilidad, etc. – pensar calidad del servicio.

Diagrama de Clases del Dominio

Conceptual Perspective
(domain model)

Raw UML class diagram notation used to visualize real-world concepts.



Specification or
Implementation
Perspective
(design class diagram)

Raw UML class diagram notation used to visualize software elements.

Usos de UML

- Diseño Conceptual – Conceptos claves del dominio del problema
- Punto de Vista de Especificación – Describe abstracciones de software o componentes con especificaciones e interfaces. El lenguaje de implementación no es considerado.
- Perspectiva de Implementación – Los diagramas describen desde el punto de vista de una tecnología como Java, etc.

Conceptos de Clases

- Clase conceptual – conceptos claves del mundo real en el espacio del problema.
- Clase de software – una especificación de un componente de software
- Clase de implementación – una clase de un lenguaje orientado a objetos especifico

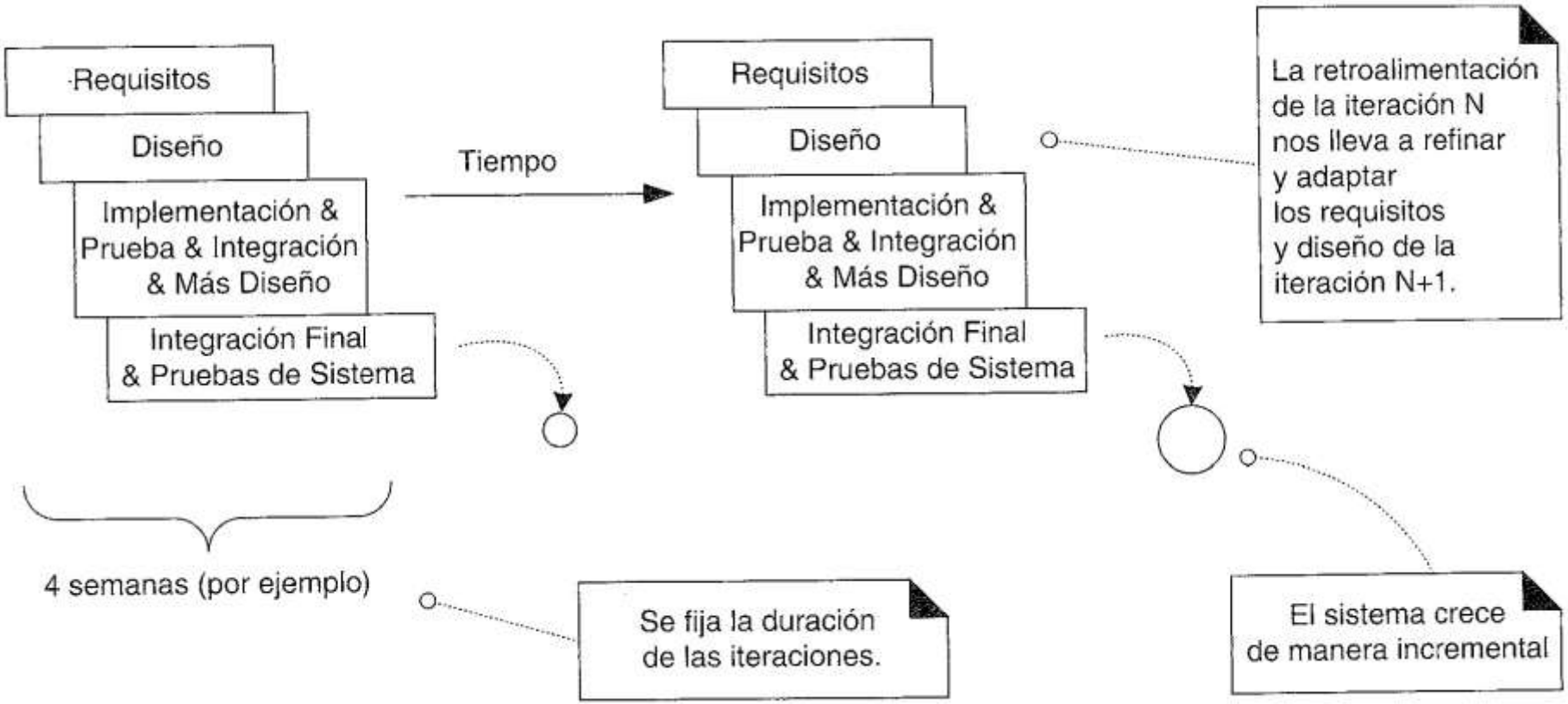
Ciclo de vida en “cascada” secuencial

- Determinar, registrar y acordar un conjunto de requisitos completo y fijo.
- Diseñar un sistema basado en estos requisitos.
- Implementar en base a ese diseño.

Desarrollo Iterativo Vs Desarrollo de Cascada

- Cascada-se completa cada etapa antes de seguir a la siguiente.
- Iterativo
  - se trabaja en ciclos pequeños que incluyen todas las etapas
  - se divide la funcionalidad del sistema en partes coherentes donde se puede aplicar todo el proceso de desarrollo

Desarrollo Iterativo e Incremental



- Mini-proyectos cortos derivados de parte de los requerimientos
- Abordar las partes de mas alto riesgo y valor para el cliente en las primeras iteraciones
- Involucrar continuamente a los usuarios para evaluación, retroalimentación y requisitos.
- Construir en las primeras iteraciones una arquitectura central solida.
- Aplicar casos de uso
- Modelar visualmente con UML

Comienzo del Proyecto

- Análisis de la factibilidad del proyecto
- Estimar el costo en tiempo y plata.
- Alcance del proyecto
- Análisis de los requerimientos no funcionales.
- Creación del caso de negocios.
  - Crear versus comprar?
- Investigación y preparación del ambiente de desarrollo.

Primeras Iteraciones

- Impulsadas por los riesgos mas salientes
  - Centrarse en la arquitectura medular del sistema.
  - Centrarse en las modules mas complejos del sistema.
- Impulsadas por la funcionalidad que mas le importa a los usuarios.
- Obtener retroalimentación de los usuarios lo antes posible.