

## 6.170 Repaso de la prueba

### Clases:

1. Desacoplamiento.
2. Abstracción de datos.
3. Funciones de abstracción e invariantes de representación.
4. Abstracción de iteración e iteradores.
5. Modelos de objeto e invariantes.
6. Igualdad, copia y vistas.
7. Análisis dinámico.
8. Patrones de diseño.
9. Subtipado.
10. Prácticas.

### Desacoplamiento

Clase 2, clase 3, capítulo 1, capítulo 13: 1-3, capítulo 2

Descomposición.  
División del trabajo.  
Reusabilidad.  
Análisis modular.  
Cambio localizado.

### Diseño *Top-Down* frente a modularización

#### Desacoplamiento

Clase 2: Usos, dependencias, especificaciones, *MDDs* (Diagrama de dependencia de módulos).

Diagramas de uso: árboles, capas y ciclos.  
Razonamiento.  
Reusabilidad.  
Orden de construcción.

- Dependencias y especificaciones: *MDDs* para:
  - Supuestos debilitados.
  - Evaluación de cambios.
  - Comunicaciones.
  - Implementaciones múltiples.

#### Desacoplamiento

Clase 2: *MDDs*, técnicas

- *MDDs*.
  - Partes de la especificación.

- Partes de la implementación.
- Satisface, depende, relaciones de dependencia débil.
- Técnicas
  - Fachada: nueva parte de la implementación entre dos conjuntos de partes.
  - Ocultar la representación: evita mencionar cómo se han representado los datos.
  - Polimorfismo: “muchas formas”.
  - *Callbacks*: referencias a un procedimiento en tiempo de ejecución.

## **Desacoplamiento**

Clase 3: *Namespace* de Java, control de acceso

- *Namespace* de Java
  - Paquetes {Interfaces, clases} {métodos, nombres de campos}
- Control de acceso
  - público: se accede desde cualquier lugar.
  - protegido: se accede desde dentro del paquete o por una subclase de fuera del mismo.
  - por defecto: se accede desde dentro del paquete.
  - privado: sólo desde dentro de la propia clase.

## **Desacoplamiento**

Clase 3: Lenguajes seguros, interfaces

- Lenguajes seguros
  - Una parte sólo debería depender de otra si ésta hace referencia a la primera.
  - Tipos fuertes: el acceso de tipo  $t$  en el programa texto, se garantiza en tiempo de ejecución.
  - Comprobación de tipos en tiempo de compilación: “tipado estático”.
- Interfaces: subtipado más flexible
  - Expresa la especificación pura.
  - Permite varias partes de la implementación en una parte de la especificación.

## **Desacoplamiento**

Clase 3: Instrumentación de un programa

- Abstracción por parametrización.
- Desacoplamiento con interfaces.
- Interfaces contra clases abstractas.

- Campos estáticos.

### **Abstracción de datos**

Clase 4, clase 5, capítulos 3-5, capítulo 9

- Especificaciones.
  - Precondición (*requires*)
    - Obligación sobre el cliente (el que invoca un método).
    - Si se omite: *true*; no requiere nada.
  - Poscondición (*effects*)
    - Obligación sobre el implementador.
    - No se puede omitir.
  - Condición estructural (*modifies*).
    - Describe qué estado pequeño se ha modificado.
    - Si se omite: no modifica nada.

### **Abstracción de datos**

Clase 4: Especificación

- Especificación operacional: serie de pasos que el método lleva a cabo.
- Especificación declarativa: no facilita detalles sobre los pasos intermedios (preferible).
- Excepciones y precondiciones (decisiones).
  - Precondiciones: coste de la comprobación, alcance del método.
  - Comprobaciones mediante aserciones en tiempo de ejecución.
  - Si se viola, lanza excepción no comprobada (no mencionada en la especificación).

### **Abstracción de datos**

Clase 4: Especificaciones

- Abreviaturas
  - *Returns*: no modifica nada y devuelve un valor.
  - *Throws*: condición y excepción, facilitadas en la cláusula *throws*; no modifica nada.
- Orden de la especificación: una especificación *A* es al menos tan fuerte como una especificación *B* si:
  - La precondición de *A* no es más fuerte que la de *B*.
  - La precondición de *A* no es más débil que la de *B*, para los estados que satisfacen la precondición de *B*.

- (Siempre es posible debilitar y fortalecer la precondición).

## **Abstracción de datos**

### Clase 4: Especificaciones

- Cómo juzgar especificaciones
  - Coherente.
  - Informativa.
  - Suficientemente fuerte.
  - Suficientemente débil.
- *Firewall* crucial entre implementador y cliente.

## **Abstracción de datos**

### Clase 5: Tipos abstractos

- Abstracción de datos: el tipo se caracteriza por las operaciones que usted puede llevar a cabo en él.
- Mutable: se puede modificar; facilita operaciones que cuando se ejecutan, hacen que los resultados de otras operaciones sobre el mismo objeto den resultados diferentes (*Vectors*).
- Inmutable: no pueden modificarse (*Strings*).

## **Abstracción de datos**

### Clase 5: Tipos abstractos

Operaciones (T = tipo abstracto, t = cualquier otro tipo)

- Constructores: T ->t
- Productores : T, t-> T
- Mutadores: T, t-> void
- Observadores: T, t-> t

Ejemplo : listas

## **Abstracción de datos**

### Clase 5: Tipos abstractos

Cómo diseñar un tipo abstracto

- Unas pocas operaciones sencillas se pueden combinar de diversas maneras.
- Las operaciones deberían tener un propósito bien definido, un comportamiento coherente.
- El conjunto de operaciones debería ser adecuado.

- El tipo puede ser genérico (lista, conjunto, grafo), o específico del dominio (mapa de calles, base de datos de empleados, agenda telefónica), pero no los dos al mismo tiempo.

## **Abstracción de datos**

Clase 5: Tipos abstractos

Representación: la clase que implementa un tipo abstracto, proporciona una representación.

Independencia de representación

- Cómo garantizar que el uso de un tipo abstracto es independiente de la representación.
- Las alteraciones en la representación no deben afectar a la utilización del código.

Exposición de representación

- La representación se pasa al cliente.
- Al cliente se le permite acceso directo a la representación.
- Es necesario una esmerada disciplina de programación.

## **Abstracción de datos**

Clase 5: Tipos abstractos

Mecanismos del lenguaje

- Campos privados: previenen el acceso a la representación.
- Interfaces: independencia de representación.
- Interfaces: independencia de representación (*List* -> *ArrayList*, *LinkedList*).
  - No están permitidos los campos estáticos.
  - No pueden tener constructores.

## **Abstracción de datos**

Clase 6: Funciones de abstracción e invariantes de representación

Invariante Rep. (*IR*)

- Restricción que caracteriza si una interfaz de un tipo de datos abstracto está bien formada (desde el punto de vista de la representación).
- *IR: Object* -> *Boolean*
- Algunas propiedades del modelo de objeto no están en el *IR* (Ej. repartos/multiplicidades).
- Algunas propiedades del *IR* no están en el modelo de objeto (Ej. primitivos).

## **Abstracción de datos**

Clase 6: Funciones de abstracción e invariantes de representación

## Razonamiento inductivo

- Invariante de representación: posibilita el razonamiento modular.
- El método constructor crea un objeto que satisface el invariante.
- El método productor mantiene el invariante.
- El método modificador mantiene el invariante Rep.
- El método observador no realiza modificaciones, por tanto, se mantiene el invariante.

### **Abstracción de datos**

Clase 6: Funciones de abstracción e invariantes de representación

Función de abstracción: interpreta la representación

- Objetos concretos: objetos reales de la implementación.
- Objetos abstractos: objetos matemáticos que se corresponden con el modo en el que la especificación del tipo abstracto define sus valores.
- Función de abstracción: función entre los dominios concretos y abstractos.
- Puede ser parcial.
- Representaciones distintas poseen diferentes funciones de abstracción.

### **Abstracción de datos**

Clase 6: Funciones de abstracción e invariantes de representación

- Efectos colaterales benevolentes: permite que los métodos observadores modifiquen la representación mientras se mantenga el valor abstracto.
- *IRs* (invariantes de representación).
  - Razonamiento modular.
  - Ayuda a identificar errores.
- Función de abstracción: especifica cómo la representación de un tipo de datos abstracto se interpreta como un valor abstracto.

### **Abstracción de datos**

Clase 7: Abstracción de iteración e iteradores

- Exposición de representación: hace que el método *remove()* lance *UnsupportedOperationException*.
- Consulte el capítulo 6 del libro de texto.

### **Modelos de objetos e invariantes**

Clase 8, capítulo 12:1

Modelo de objeto: descripción de una colección de configuraciones.

- Clasificación de objetos.
- Relaciones entre objetos.
- Subconjuntos (implementa, extiende).

- Relaciones y etiquetas.
- Multiplicidad: cómo muchos objetos de una clase pueden estar relacionados con un determinado objeto de otra.
- Mutabilidad: cómo pueden modificarse los estados.

## Modelos de objetos e invariantes

- Símbolos de multiplicidad:
  - ( $\geq 0$ )
  - + ( $\geq 1$ )
  - ? (0 ó 1)
  - ! (exactamente 1)
- Origen -> destino
  - Final de la flecha: ¿cuántos destinos están asociados con cada origen?
  - Inicio de la flecha: ¿cuántos orígenes se pueden asociar a un destino?
- Diagramas de instancia.

## Modelos de objetos e invariantes

- Modelos de objetos de programa.
- Puntos de vista concretos y abstractos.
  - Función de abstracción: muestra cómo los valores concretos se interpretan como abstractos.
  - Invariante de representación: el modelo de objeto es un tipo de invariante de representación –una restricción que se mantiene durante el tiempo de vida de un programa.
  - Exposición de representación: un tipo de datos abstracto suministra acceso directo a uno de los objetos dentro del contorno del invariante de representación.

## Igualdad, copia y vistas

Clase 9, Capítulos 5-7

- El contrato de la clase *Object*.
  - *equals()*
  - *hashCode()*
- Propiedades de igualdad (*Point* y *ColorPoint*).
  - Reflexividad.
  - Simetría.
  - Transitividad.
- *Hashing*: si dos objetos son *equals()* -> deben tener el mismo *hashCode()*.

## Igualdad, copia y vistas

- Copia
  - Superficial: los campos apuntan a los mismos campos que el objeto original.
  - Profunda.
- Interfaz clonable.
- Igualdad entre elementos y contenedores.
  - La solución de Liskov:
    - *Equals* – equivalencia desde el punto de vista del comportamiento.
    - *Similar* – equivalencia desde el punto de vista observacional.

## Igualdad, copia y vistas

- Exposición de representación: el contorno incluye la clase de los elementos (*LinkedList*, por ejemplo).
  - Mutación de las claves *hash*.
- Vistas
  - Objetos diferentes que ofrecen distinto tipo de acceso a la estructura de datos subyacente.
  - Tanto la vista como la estructura subyacente son modificables.

## Análisis dinámico

Clase 10, clase 11, capítulo 10

- Cómo ejecutar un programa y observar su comportamiento.
- Dijkstra: “*Las pruebas pueden revelar la presencia de errores pero nunca su ausencia*”.
- No se puede depender sólo del análisis dinámico –se necesitan buenas especificaciones y el diseño.

## Análisis dinámico

Clase 10: Programación defensiva

- Directrices
  - Inserción de comprobaciones redundantes: aserciones en tiempo de ejecución.
  - Mientras escribe el código.
  - ¿Dónde?
    - Al inicio de un procedimiento (Precondición).
    - Al final de un procedimiento complicado (Postcondición).
    - Cuando una operación puede producir un efecto externo.

## Análisis dinámico

Clase 10: Programación defensiva

- Cómo detectar excepciones comunes.
  - *NullPointerException*
  - *ArrayIndexOutOfBoundsException*
  - *ClassCastException*
- Comprobación del invariante de representación.
  - *public void repCheck()* lanza (expn en tiempo de ejecución).
- Marco de certificación.
  - *public static void assert* (booleano b, *String loc*)
  - *Assert.assert(..., "MyClass.myMethod")*;

### **Análisis dinámico**

#### Clase 10: Programación defensiva

- Aserciones en subclases.
- Cómo responder al fallo.
  - Reparar: complicado, más errores, si usted conoce la causa → ¿podría haberla evitado?
  - Ejecutar acciones especiales: depende del sistema → es difícil definir un conjunto de acciones.
  - Abandonar ejecución: depende del programa; compilador contra procesador de texto.

### **Análisis dinámico**

#### Clase 11: Pruebas

- Algunas consideraciones en relación a las pruebas
  - Propiedades que usted quiere comprobar (dominio del problema, conocimiento del programa).
  - Módulos que usted quiere probar (críticos, complejos, proclives a un mal funcionamiento).
  - Cómo generar casos de prueba.
  - Cómo comprobar los resultados.
  - Cuándo sabe si ha acabado.

### **Análisis dinámico**

#### Clase 11: Pruebas de regresión

- *Suites* de prueba que se pueden volver a ejecutar.
- Programación *test-first*: construcción de pruebas de regresión antes de que el código de la aplicación sea escrito (parte de la programación extrema).

### **Análisis dinámico**

## Clase 11: Criterios

- $S(t, P(t)) = false$ ;  $t$  es un caso de prueba fallido.
- $C$ : *Suite*, programa, especificación  $\rightarrow$  booleano.
- $C$ : *Suite*, especificación  $\rightarrow$  el valor booleano es un criterio basado en la especificación; caja negra.
- $C$ : *Suite*, programa  $\rightarrow$  el valor booleano es un criterio basado en el programa; caja de cristal.

### **Análisis dinámico**

#### Clase 11: Subdominios

- Subdominios: divisiones de espacio de datos de entrada.
  - Determinan si los *suites* de prueba son lo suficientemente buenos.
  - Guían la actividad de pruebas hacia las regiones más propensas a los errores.
- Subdominios reveladores.

### **Análisis dinámico**

#### Clase 11: Criterios de subdominio

- Cobertura de sentencia: toda sentencia debe ejecutarse al menos una vez.
- Cobertura de decisión: todas las extremidades del grafo de control de flujo se deben ejecutar.
- Cobertura de condición: las expresiones booleanas deben evaluarse como *true* y *false*; MCDC.
- Pruebas extremas: casos extremos para cada expresión condicional.
- Criterios basados en la especificación: únicamente en función de subdominios.
  - Conjunto vacío, no vacío y contiene elemento, no vacío y no contiene elemento.

### **Análisis dinámico**

#### Clase 11: Viabilidad y factibilidad

- Un criterio es viable si es posible cumplirlo.
- Utilizar criterios basados en la especificación para guiar el desarrollo del *suite* de prueba.
- Criterios basados en el programa para evaluar este *suite* (cobertura de cantidad de código).

### **Patrones de diseño**

#### Clase 12, clase 13, clase 14, capítulo 15

- Hasta ahora:
  - Encapsulación (ocultar la representación de datos).
  - Subclases (herencia).

- Iteración.
- Excepciones.
- No usar los patrones de diseño antes de tiempo.
- La complejidad disminuye la capacidad de comprensión.

## Patrones de diseño

### Clase 12: Patrones de creación

- *Factories*
  - Método *factory*: método que fabrica un objeto de un tipo en concreto.
  - Objeto *factory*: objeto que encapsula métodos de tipo *factory*.
  - Prototipo: el objeto puede clonarse a sí mismo (método *clone()*), objeto pasado a través de un método (en vez de un objeto *factory*).

## Patrones de diseño

### Clase 12: Patrones de creación

- Reparto
  - *Singleton*: existe sólo un objeto de una clase.
  - *Interning*: reutiliza un objeto en vez de crear otros nuevos; sólo es apropiado para objetos inmutables.
  - *Flyweight*: (generalización de *interning*), puede utilizarse si la mayor parte del objeto es inmutable.
    - Estados intrínsecos contra estados extrínsecos.
    - Sólo se utiliza si el espacio es un obstáculo crítico.

## Patrones de diseño

### Clase 13: Patrones de comportamiento

- Comunicación multi-modo.
  - *Observer*: mantiene una lista de *Observers* (que siguen una interfaz en concreto) que debe notificarse cuando se den cambios de estado; requiere los métodos observadores *add* y *remove*.
  - *Blackboard*: (generalización del patrón *Observer*); múltiples fuentes de datos y múltiples visualizadores; asíncrono.
    - Depósito de mensajes que puede ser leído y escrito por todos los procesos.
    - Interoperabilidad; formato de mensaje bien comprendido.
  - *Mediator*: (intermediario entre *Observer* y *Blackboard*); desacopla información, peor no controla, síncrono.

## Patrones de diseño

### Clase 13: Compuestos (objetos compuestos) transversales

- Admite muchas operaciones distintas.

- Realiza operaciones en las subpartes de un compuesto.
- *Interpreter*: agrupa operaciones de un tipo especial de objeto.
- *Procedural*: agrupa todo el código que implementa una operación en concreto.
- *Visitor*: recorre en profundidad una estructura jerárquica; *Nodes* aceptan *Visitors*; *Visitors* visitan *Nodes*.

### Patrones de diseño

Clase 14: Patrones estructurales

- *Wrappers*

Patrón	Funcionalidad	Interfaz
• <i>Adaptor</i> (Interoperabilidad)	Igual	Diferente
• <i>Decorator</i> (extiende)	Diferente	Igual
• <i>Proxy</i> (controla o limita)	Igual	Igual

### Patrones de diseño

Clase 14: Patrones estructurales

- Implementación de envoltorios.
  - Subclases.
  - Delegación: almacena un objeto en un campo; implementación más adecuada para envoltorios;
- Compuestos.
  - Permiten que un cliente controle una unidad o colección de unidades de la misma forma.

### Subtipado

Clase 15, capítulo 7

- Diagramas de dependencia de módulos.
- Principio de la sustitución.
  - Firmas.
  - Métodos.
    - Requiere menos/contravarianza.
    - Garantiza más/covarianza.
  - Propiedades.
- Subclases de Java frente a subtipos.
- Interfaz.
  - Garantiza el comportamiento con aquellos que comparten el código.
  - Herencia múltiple.

### Prácticas: API de colecciones de Java

- La jerarquía de tipos.
  - Interfaces: *Collection*, *Set*, *SortedSet*, *List*
  - Implementaciones basadas en estructuras jerárquicas esqueléticas: *AbstractCollection*, *AbstractSet*, *AbstractList*, *AbstractSequentialList*.
  - Implementaciones concretas: *TreeSet*, *HashSet*, *ArrayList*, *LinkedList*.
- Estructura paralela.
- Interfaces frente a clases abstractas.

## Prácticas: API de colecciones de Java

Clase 16, capítulo 13, capítulo 14

- Métodos opcionales: *throws UnsupportedOperationException*.
- Polimorfismo.
- Implementaciones basadas en jerarquías esqueléticas (“métodos *template*” y “*métodos hook*”).
- Capacidad, asignación, *garbage collector*.
- Copias, conversiones, *wrappers*.
- Colecciones ordenadas: *Comparable* frente a *Comparator*.
- Vistas.

## Prácticas: JUnit

Clase 17

- Diagrama de dependencia de módulos: totalmente conectado.
- Patrones de diseño.
  - Método *Template*.
  - *Command*.
  - *Composite*.
  - *Observer*.
- *Suite* de prueba utilizando Reflexión de Java.

## Prácticas: Tagger

Clase 18

- Aspectos del diseño.
  - Acciones.
  - Referencias cruzadas.
  - Mapas de propiedades.
  - Numeración automática.
  - Vistas de plantillas de estilo.
  - Enumeraciones *Type-Safe*.
- Requisitos de calidad.
- Densidad del patrón.

## Modelos de objetos conceptuales

## Clase 19, capítulo 11-12

- Indivisible.
- Inmutable.
- No interpretable.
- Conjunto: colección de átomos
  - Dominios: conjuntos sin superconjuntos.
  - Relación: relativa a los átomos.
  - Transposición: ~relación.
  - Cierre transitivo: +relación.
- Cierre reflexivo: \*relación.

## Modelos de objetos conceptuales

- Relaciones ternarias
- Relaciones indexadas
- Ejemplos:
  - Tipos de Java: *Object*, *Var*, *Type*.
  - Meta modelo: notación gráfica del modelado de objeto.
  - Numeración: *Tagger*.

## Estrategias de diseño

### Clase 20

- Proceso de desarrollo.
  - Análisis del programa (modelos de objeto y operaciones).
  - Diseño (codificar el modelo de objeto, diagrama de dependencia de módulos, especificaciones del módulo).
  - Implementación.
- Pruebas
  - Tests de regresión.
  - Aserciones en tiempo de ejecución.
  - Invariantes de representación.

## Estrategias de diseño

### Propiedades del diseño

- Extensibilidad o capacidad de extensión.
  - Suficiencia sobre el modelo de objeto.
  - Localidad y desacoplamiento.
- Fiabilidad.
  - Modelado cuidadoso.
  - Revisión, análisis, pruebas.
- Eficacia.
  - Modelo de objeto.
  - Evite la parcialidad.

- Optimización.
- Escoja las representaciones.

### **Estrategias de diseño**

- Transformaciones en el modelo de objeto.
- Introducción de una generalización (subconjuntos).
- Inserción de una colección.
- Inversión de una relación.
- Desplazamiento de una relación.
- Tabla relacional
- Adición de un estado redundante.
- Descomposición de relaciones mutables.
- Interpolación de una interfaz.
- Eliminación de conjuntos dinámicos.