Paradigmas Fundamentales de Programación

Introducción al modelo con estado

Juan Francisco Díaz Frias

Maestría en Ingeniería, Énfasis en Ingeniería de Sistemas y Computación
Escuela de Ingeniería de Sistemas y Computación,
home page: http://eisc.univalle.edu.co
Universidad del Valle - Cali, Colombia

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

1 ¿Qué es estado?
- Estado y Estado implícito
- Estado y Estado explícito

2 Diseño de sistemas y estado
- El principio de abstracción
- ¿Cómo soportar el principio de abstracción?

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

1 ¿Qué es estado?
- Estado y Estado implícito
- Estado y Estado explícito

2 Diseño de sistemas y estado
- El principio de abstracción
- ¿Cómo soportar el principio de abstracción?

Modelos y Paradigmas de Programación

Universidad del Valle

Plan

1 ¿Qué es estado?
- Estado y Estado implícito
- Estado y Estado explícito

2 Diseño de sistemas y estado
- El principio de abstracción
- ¿Cómo soportar el principio de abstracción?

¿Qué es estado? (1)

Definición

Un estado es una secuencia de valores en el tiempo que contienen los resultados intermedios de una computación específica. Puede ser: implícito (declarativo) o explícito.

Estado implícito (declarativo)

- Existe en la mente del programador.
- No se necesita soporte por parte del modelo de computación.

Estado implícito: ejemplo

```
fun {SumList Xs S}
   case Xs
   of nil then S
   [] X|Xr then
      {SumList Xr X+S}
   end
end
```

¿Qué es estado? (1)

Definición

Un estado es una secuencia de valores en el tiempo que contienen los resultados intermedios de una computación específica. Puede ser: implícito (declarativo) o explícito.

Estado implícito (declarativo)

- Existe en la mente del programador.
- No se necesita soporte por parte del modelo de computación.

Estado implícito: ejemplo

```
fun {SumList Xs S}
   case Xs
   of nil then S
   [] X|Xr then
      {SumList Xr X+S}
   end
end
```

- Observe la pareja (Xs#S)
- En {SumList [1 2 3 4] 0}:
  - ([1 2 3 4]#0)
  - ([2 3 4]#1) ([3 4]#3)
  - ([4]#6) (nil#10)
- SumList calcula con estado, aunque en el programa ni el modelo de computación lo saben.

¿Qué es estado? (1)

Definición

Un estado es una secuencia de valores en el tiempo que contienen los resultados intermedios de una computación específica. Puede ser: implícito (declarativo) o explícito.

Estado implícito (declarativo)

- Existe en la mente del programador.
- No se necesita soporte por parte del modelo de computación.

Estado implícito: ejemplo

```
fun {SumList Xs S}
   case Xs
   of nil then S
   [] X|Xr then
      {SumList Xr X+S}
   end
end
```

- Observe la pareja `(Xs#S)`
- En `{SumList [1 2 3 4] 0}`:
  `([1 2 3 4]#0)`
  `([2 3 4]#1)` `([3 4]#3)`
  `([4]#6)` `(nil#10)`
- `SumList` calcula con estado, aunque ni el programa ni el modelo de computación "lo saben".

¿Qué es estado? (1)

Definición

Un estado es una secuencia de valores en el tiempo que contienen los resultados intermedios de una computación específica. Puede ser: implícito (declarativo) o explícito.

Estado implícito (declarativo)

- Existe en la mente del programador.
- No se necesita soporte por parte del modelo de computación.

Estado implícito: ejemplo

```
fun {SumList Xs S}
   case Xs
   of nil then S
   [] X|Xr then
      {SumList Xr X+S}
   end
end
```

- Observe la pareja `(Xs#S)`
- En `{SumList [1 2 3 4] 0}`:

```
([1 2 3 4]#0)
([2 3 4]#1) ([3 4]#3)
([4]#6)     (nil#10)
```

- `SumList` calcula con estado, aunque ni el programa ni el modelo de computación "lo saben".

¿Qué es estado? (1)

Definición

Un estado es una secuencia de valores en el tiempo que contienen los resultados intermedios de una computación específica. Puede ser: implícito (declarativo) o explícito.

Estado implícito (declarativo)

- Existe en la mente del programador.
- No se necesita soporte por parte del modelo de computación.

Estado implícito: ejemplo

```
fun {SumList Xs S}
   case Xs
   of nil then S
   [] X|Xr then
      {SumList Xr X+S}
   end
end
```

- Observe la pareja `(Xs#S)`
- En `{SumList [1 2 3 4] 0}`:

```
([1 2 3 4]#0)
([2 3 4]#1) ([3 4]#3)
([4]#6)     (nil#10)
```

- `SumList` calcula con estado, aunque ni el programa ni el modelo de computación "lo saben".

Plan

1 ¿Qué es estado?
- Estado y Estado implícito
- Estado y Estado explícito

2 Diseño de sistemas y estado
- El principio de abstracción
- ¿Cómo soportar el principio de abstracción?

¿Qué es estado? (2)

Estado explícito en un procedimiento . . .

es un estado cuyo tiempo de vida se extiende más allá de una invocación al procedimiento sin estar presente en los argumentos del procedimiento.

Motivación

Extender una funcion sin cambiar su interfaz. Por ejemplo, extender `SumList` para contar cuántas veces es invocada.

- El estado explícito no se puede expresar en el modelo declarativo.
- Extendemos el modelo con una especie de contenedor que denominamos una celda.

`SumList` con celdas

```
local
   C={NewCell 0}
in
   fun {SumList Xs S}
      C:=@C+1
      case Xs
      of nil then S
      [] X|Xr then
         {SumList Xr X+S}
      end
   end
   fun {ContadorSum} @C end
end
```

¿Qué es estado? (2)

Estado explícito en un procedimiento ...

es un estado cuyo tiempo de vida se extiende más allá de una invocación al procedimiento sin estar presente en los argumentos del procedimiento.

Motivación

Extender una funcion sin cambiar su interfaz. Por ejemplo, extender `SumList` para contar cuántas veces es invocada.

- El estado explícito no se puede expresar en el modelo declarativo.
- Extendemos el modelo con una especie de contenedor que denominamos una celda.
- Una celda tiene un nombre, un tiempo de vida indefinido, y un contenido que puede ser modificado.
- Modelo con estado: Declarativo + Celdas
- El estado es visible tanto en el programa como en el modelo de computación.

`SumList` con celdas

```
local
   C={NewCell 0}
in
   fun {SumList Xs S}
      C:=@C+1
      case Xs
      of nil then S
      [] X|Xr then
         {SumList Xr X+S}
      end
   end
   fun {ContadorSum} @C end
end
```

Universidad del Valle

¿Qué es estado? (2)

Estado explícito en un procedimiento ...

es un estado cuyo tiempo de vida se extiende más allá de una invocación al procedimiento sin estar presente en los argumentos del procedimiento.

Motivación

Extender una funcion sin cambiar su interfaz. Por ejemplo, extender `SumList` para contar cuántas veces es invocada.

- El estado explícito no se puede expresar en el modelo declarativo.
- Extendemos el modelo con una especie de contenedor que denominamos una celda.
- Una celda tiene un nombre, un tiempo de vida indefinido, y un contenido que puede ser modificado.
- Modelo con estado: Declarativo + Celdas
- El estado es visible tanto en el programa como en el modelo de computación.

`SumList` con celdas

```
local
   C={NewCell 0}
in
   fun {SumList Xs S}
      C:=@C+1
      case Xs
      of nil then S
      [] X|Xr then
         {SumList Xr X+S}
      end
   end
   fun {ContadorSum} @C end
end
```

¿Qué es estado? (2)

Estado explícito en un procedimiento ...

es un estado cuyo tiempo de vida se extiende más allá de una invocación al procedimiento sin estar presente en los argumentos del procedimiento.

Motivación

Extender una funcion sin cambiar su interfaz. Por ejemplo, extender `SumList` para contar cuántas veces es invocada.

- El estado explícito no se puede expresar en el modelo declarativo.
- Extendemos el modelo con una especie de contenedor que denominamos una celda.
- Una celda tiene un nombre, un tiempo de vida indefinido, y un contenido que puede ser modificado.
- Modelo con estado: Declarativo + Celdas
- El estado es visible tanto en el programa como en el modelo de computación.

`SumList` con celdas

```
local
   C={NewCell 0}
in
   fun {SumList Xs S}
      C:=@C+1
      case Xs
      of nil then S
      [] X|Xr then
         {SumList Xr X+S}
      end
   end
   fun {ContadorSum} @C end
end
```

¿Qué es estado? (2)

Estado explícito en un procedimiento ...

es un estado cuyo tiempo de vida se extiende más allá de una invocación al procedimiento sin estar presente en los argumentos del procedimiento.

Motivación

Extender una funcion sin cambiar su interfaz. Por ejemplo, extender `SumList` para contar cuántas veces es invocada.

- El estado explícito no se puede expresar en el modelo declarativo.
- Extendemos el modelo con una especie de contenedor que denominamos una celda.
- Una celda tiene un nombre, un tiempo de vida indefinido, y un contenido que puede ser modificado.
- Modelo con estado: Declarativo + Celdas
- El estado es visible tanto en el programa como en el modelo de computación.

`SumList` con celdas

```
local
   C={NewCell 0}
in
   fun {SumList Xs S}
      C:=@C+1
      case Xs
      of nil then S
      [] X|Xr then
         {SumList Xr X+S}
      end
   end
   fun {ContadorSum} @C end
end
```

¿Qué es estado? (2)

Estado explícito en un procedimiento ...

es un estado cuyo tiempo de vida se extiende más allá de una invocación al procedimiento sin estar presente en los argumentos del procedimiento.

Motivación

Extender una funcion sin cambiar su interfaz. Por ejemplo, extender `SumList` para contar cuántas veces es invocada.

- El estado explícito no se puede expresar en el modelo declarativo.
- Extendemos el modelo con una especie de contenedor que denominamos una celda.
- Una celda tiene un nombre, un tiempo de vida indefinido, y un contenido que puede ser modificado.
- Modelo con estado: Declarativo + Celdas
- El estado es visible tanto en el programa como en el modelo de computación.

`SumList` con celdas

```
local
   C={NewCell 0}
in
   fun {SumList Xs S}
      C:=@C+1
      case Xs
      of nil then S
      [] X|Xr then
         {SumList Xr X+S}
      end
   end
   fun {ContadorSum} @C end
end
```

¿Qué es estado? (2)

Estado explícito en un procedimiento ...

es un estado cuyo tiempo de vida se extiende más allá de una invocación al procedimiento sin estar presente en los argumentos del procedimiento.

Motivación

Extender una funcion sin cambiar su interfaz. Por ejemplo, extender `SumList` para contar cuántas veces es invocada.

- El estado explícito no se puede expresar en el modelo declarativo.
- Extendemos el modelo con una especie de contenedor que denominamos una celda.
- Una celda tiene un nombre, un tiempo de vida indefinido, y un contenido que puede ser modificado.
- Modelo con estado: Declarativo + Celdas
- El estado es visible tanto en el programa como en el modelo de computación.

`SumList` con celdas

```
local
   C={NewCell 0}
in
   fun {SumList Xs S}
      C:=@C+1
      case Xs
      of nil then S
      [] X|Xr then
         {SumList Xr X+S}
      end
   end
   fun {ContadorSum} @C end
end
```

- Operaciones nuevas: `NewCell`, `@`, `:=`
- Estado encapsulado

¿Qué es estado? (2)

Estado explícito en un procedimiento . . .

es un estado cuyo tiempo de vida se extiende más allá de una invocación al procedimiento sin estar presente en los argumentos del procedimiento.

Motivación

Extender una funcion sin cambiar su interfaz. Por ejemplo, extender `SumList` para contar cuántas veces es invocada.

- El estado explícito no se puede expresar en el modelo declarativo.
- Extendemos el modelo con una especie de contenedor que denominamos una celda.
- Una celda tiene un nombre, un tiempo de vida indefinido, y un contenido que puede ser modificado.
- Modelo con estado: Declarativo + Celdas
- El estado es visible tanto en el programa como en el modelo de computación.

`SumList` con celdas

```
local
   C={NewCell 0}
in
   fun {SumList Xs S}
      C:=@C+1
      case Xs
      of nil then S
      [] X|Xr then
         {SumList Xr X+S}
      end
   end
   fun {ContadorSum} @C end
end
```

- Operaciones nuevas: `NewCell`, `@` y `:=`.
- Estado encapsulado.

¿Qué es estado? (2)

Estado explícito en un procedimiento . . .

es un estado cuyo tiempo de vida se extiende más allá de una invocación al procedimiento sin estar presente en los argumentos del procedimiento.

Motivación

Extender una funcion sin cambiar su interfaz. Por ejemplo, extender `SumList` para contar cuántas veces es invocada.

- El estado explícito no se puede expresar en el modelo declarativo.
- Extendemos el modelo con una especie de contenedor que denominamos una celda.
- Una celda tiene un nombre, un tiempo de vida indefinido, y un contenido que puede ser modificado.
- Modelo con estado: Declarativo + Celdas
- El estado es visible tanto en el programa como en el modelo de computación.

`SumList` con celdas

```
local
   C={NewCell 0}
in
   fun {SumList Xs S}
      C:=@C+1
      case Xs
      of nil then S
      [] X|Xr then
         {SumList Xr X+S}
      end
   end
   fun {ContadorSum} @C end
end
```

- Operaciones nuevas:`NewCell`, `@` y `:=`.
- Estado encapsulado.

¿Qué es estado? (2)

Estado explícito en un procedimiento . . .

es un estado cuyo tiempo de vida se extiende más allá de una invocación al procedimiento sin estar presente en los argumentos del procedimiento.

Motivación

Extender una funcion sin cambiar su interfaz. Por ejemplo, extender `SumList` para contar cuántas veces es invocada.

- El estado explícito no se puede expresar en el modelo declarativo.
- Extendemos el modelo con una especie de contenedor que denominamos una celda.
- Una celda tiene un nombre, un tiempo de vida indefinido, y un contenido que puede ser modificado.
- Modelo con estado: Declarativo + Celdas
- El estado es visible tanto en el programa como en el modelo de computación.

`SumList` con celdas

```
local
   C={NewCell 0}
in
   fun {SumList Xs S}
      C:=@C+1
      case Xs
      of nil then S
      [] X|Xr then
         {SumList Xr X+S}
      end
   end
   fun {ContadorSum} @C end
end
```

- Operaciones nuevas:`NewCell`, `@` y `:=`.
- Estado encapsulado.

Plan

1 ¿Qué es estado?
- Estado y Estado implícito
- Estado y Estado explícito

2 Diseño de sistemas y estado
- El principio de abstracción
- ¿Cómo soportar el principio de abstracción?

Diseño de sistemas y estado (1)

El principio de abstracción en la construcción de sistemas

- Sistema = especificación + implementación.
- Especificación: contrato que define cómo se debe comportar el sistema.
- Un sistema es correcto si su comportamiento real es acorde con el contrato. Sino, el sistema falla.
- Especificación: define interacción del resto del mundo con el sistema, desde el exterior.
- Implementación: cómo está construido el sistema, visto desde su interior.
- Sistema: serie concéntrica de capas. En cada capa, se construye una implementación que toma la siguiente especificación de más bajo nivel y provee la siguiente especificación de más alto nivel.

¿Soporta la prog. declarativa este principio de abstracción?

Diseño de sistemas y estado (1)

El principio de abstracción en la construcción de sistemas

- Sistema = especificación + implementación.
- Especificación: contrato que define cómo se debe comportar el sistema.
- Un sistema es correcto si su comportamiento real es acorde con el contrato. Sino, el sistema falla.
- Especificación: define interacción del resto del mundo con el sistema, desde el exterior.
- Implementación: cómo está construido el sistema, visto desde su interior.
- Sistema: serie concéntrica de capas. En cada capa, se construye una implementación que toma la siguiente especificación de más bajo nivel y provee la siguiente especificación de más alto nivel.

¿Soporta la prog. declarativa este principio de abstracción?

- Modelo declarativo: todo lo que el sistema "sabe" está en su exterior, los conocimientos que nacen con el sistema.
- Los procedimientos no tienen estado: todo su conocimiento está en sus argumentos.
- Entre más inteligente sea el procedimiento, los argumentos se vuelven más numerosos y más "pesados".
- Conclusión: el principio de abstracción no está completamente soportado por la programación declarativa, porque no podemos agregar conocimiento nuevo dentro de un componente.

Diseño de sistemas y estado (1)

El principio de abstracción en la construcción de sistemas

- Sistema = especificación + implementación.
- Especificación: contrato que define cómo se debe comportar el sistema.
- Un sistema es correcto si su comportamiento real es acorde con el contrato. Sino, el sistema falla.
- Especificación: define interacción del resto del mundo con el sistema, desde el exterior.
- Implementación: cómo está construido el sistema, visto desde su interior.
- Sistema: serie concéntrica de capas. En cada capa, se construye una implementación que toma la siguiente especificación de más bajo nivel y provee la siguiente especificación de más alto nivel.

¿Soporta la prog. declarativa este principio de abstracción?

- Modelo declarativo: todo lo que el sistema "sabe" está en su exterior, los conocimientos que nacen con el sistema.
- Los procedimientos no tienen estado: todo su conocimiento está en sus argumentos.
- Entre más inteligente sea el procedimiento, los argumentos se vuelven más numerosos y más "pesados".
- Conclusión: el principio de abstracción no está completamente soportado por la programación declarativa, porque no podemos agregar conocimiento nuevo dentro de un componente.
- Una alternativa: agregar estado explícito sin concurrencia.

Diseño de sistemas y estado (1)

El principio de abstracción en la construcción de sistemas

- Sistema = especificación + implementación.
- Especificación: contrato que define cómo se debe comportar el sistema.
- Un sistema es correcto si su comportamiento real es acorde con el contrato. Sino, el sistema falla.
- Especificación: define interacción del resto del mundo con el sistema, desde el exterior.
- Implementación: cómo está construido el sistema, visto desde su interior.
- Sistema: serie concéntrica de capas. En cada capa, se construye una implementación que toma la siguiente especificación de más bajo nivel y provee la siguiente especificación de más alto nivel.

¿Soporta la prog. declarativa este principio de abstracción?

- Modelo declarativo: todo lo que el sistema "sabe" está en su exterior, los conocimientos que nacen con el sistema.
- Los procedimientos no tienen estado: todo su conocimiento está en sus argumentos.
- Entre más inteligente sea el procedimiento, los argumentos se vuelven más numerosos y más "pesados".
- Conclusión: el principio de abstracción no está completamente soportado por la programación declarativa, porque no podemos agregar conocimiento nuevo dentro de un componente.
- Una alternativa: agregar estado explícito sin concurrencia.

Diseño de sistemas y estado (1)

El principio de abstracción en la construcción de sistemas

- Sistema = especificación + implementación.
- Especificación: contrato que define cómo se debe comportar el sistema.
- Un sistema es correcto si su comportamiento real es acorde con el contrato. Sino, el sistema falla.
- Especificación: define interacción del resto del mundo con el sistema, desde el exterior.
- Implementación: cómo está construido el sistema, visto desde su interior.
- Sistema: serie concéntrica de capas. En cada capa, se construye una implementación que toma la siguiente especificación de más bajo nivel y provee la siguiente especificación de más alto nivel.

¿Soporta la prog. declarativa este principio de abstracción?

- Modelo declarativo: todo lo que el sistema “sabe” está en su exterior, los conocimientos que nacen con el sistema.
- Los procedimientos no tienen estado: todo su conocimiento está en sus argumentos.
- Entre más inteligente sea el procedimiento, los argumentos se vuelven más numerosos y más “pesados”.
- Conclusión: el principio de abstracción no está completamente soportado por la programación declarativa, porque no podemos agregar conocimiento nuevo dentro de un componente.
- Una alternativa: agregar estado explícito sin concurrencia.

Diseño de sistemas y estado (1)

El principio de abstracción en la construcción de sistemas

- Sistema = especificación + implementación.
- Especificación: contrato que define cómo se debe comportar el sistema.
- Un sistema es correcto si su comportamiento real es acorde con el contrato. Sino, el sistema falla.
- Especificación: define interacción del resto del mundo con el sistema, desde el exterior.
- Implementación: cómo está construido el sistema, visto desde su interior.
- Sistema: serie concéntrica de capas. En cada capa, se construye una implementación que toma la siguiente especificación de más bajo nivel y provee la siguiente especificación de más alto nivel.

¿Soporta la prog. declarativa este principio de abstracción?

- Modelo declarativo: todo lo que el sistema “sabe” está en su exterior, los conocimientos que nacen con el sistema.
- Los procedimientos no tienen estado: todo su conocimiento está en sus argumentos.
- Entre más inteligente sea el procedimiento, los argumentos se vuelven más numerosos y más “pesados”.
- Conclusión: el principio de abstracción no está completamente soportado por la programación declarativa, porque no podemos agregar conocimiento nuevo dentro de un componente.
- Una alternativa: agregar estado explícito sin concurrencia.

Plan

1 ¿Qué es estado?
- Estado y Estado implícito
- Estado y Estado explícito

2 Diseño de sistemas y estado
- El principio de abstracción
- ¿Cómo soportar el principio de abstracción?

Diseño de sistemas y estado (2)

Propiedades para soportar el principio de abstracción

- Encapsulación: Debería ser posible ocultar lo interno de una parte.
- Composicionalidad: Debería ser posible combinar partes para construir nuevas partes.
- Instanciación/invocación: Debería ser posible crear muchas instancias de una parte basados en una sola definición.
- Extensionalidad: Debería ser posible extender una parte sin afectar a otras partes que la usen.

Diseño de sistemas y estado (2)

Propiedades para soportar el principio de abstracción

- Encapsulación: Debería ser posible ocultar lo interno de una parte.
- Composicionalidad: Debería ser posible combinar partes para construir nuevas partes.
- Instanciación/invocación: Debería ser posible crear muchas instancias de una parte basados en una sola definición.
- Extensionalidad: Debería ser posible extender una parte sin afectar a otras partes que la usen.

Diseño de sistemas y estado (2)

Propiedades para soportar el principio de abstracción

- Encapsulación: Debería ser posible ocultar lo interno de una parte.
- Composicionalidad: Debería ser posible combinar partes para construir nuevas partes.
- Instanciación/invocación: Debería ser posible crear muchas instancias de una parte basados en una sola definición.
- Extensionalidad: Debería ser posible extender una parte sin afectar a otras partes que la usen.

Diseño de sistemas y estado (2)

Propiedades para soportar el principio de abstracción

- Encapsulación: Debería ser posible ocultar lo interno de una parte.
- Composicionalidad: Debería ser posible combinar partes para construir nuevas partes.
- Instanciación/invocación: Debería ser posible crear muchas instancias de una parte basados en una sola definición.
- Extensionalidad: Debería ser posible extender una parte sin afectar a otras partes que la usen.

Conceptos para soportar esas propiedades

- El alcance léxico soporta la encapsulación.
- La programación de alto orden soporta la instanciacion y la composicionalidad.
- El estado explícito soporta la extensionalidad.

Cuidados a tener con el estado explícito

Diseño de sistemas y estado (2)

Propiedades para soportar el principio de abstracción

- Encapsulación: Debería ser posible ocultar lo interno de una parte.
- Composicionalidad: Debería ser posible combinar partes para construir nuevas partes.
- Instanciación/invocación: Debería ser posible crear muchas instancias de una parte basados en una sola definición.
- Extensionalidad: Debería ser posible extender una parte sin afectar a otras partes que la usen.

Conceptos para soportar esas propiedades

- El alcance léxico soporta la encapsulación.
- La programación de alto orden soporta la instanciación y la composicionalidad.
- El estado explícito soporta la extensionalidad.

Cuidados a tener con el estado explícito

Diseño de sistemas y estado (2)

Propiedades para soportar el principio de abstracción

- Encapsulación: Debería ser posible ocultar lo interno de una parte.
- Composicionalidad: Debería ser posible combinar partes para construir nuevas partes.
- Instanciación/invocación: Debería ser posible crear muchas instancias de una parte basados en una sola definición.
- Extensionalidad: Debería ser posible extender una parte sin afectar a otras partes que la usen.

Conceptos para soportar esas propiedades

- El alcance léxico soporta la encapsulación.
- La programación de alto orden soporta la instanciación y la composicionalidad.
- El estado explícito soporta la extensionalidad.

Cuidados a tener con el estado explícito

- Con estado, aparecen los efectos de borde globales, que dificultan razonar sobre los programas.
- Con la concurrencia, los efectos colaterales pueden ser observados y se rompen propiedades bien definidas, ocasionando errores muy difíciles de depurar.

Diseño de sistemas y estado (2)

Propiedades para soportar el principio de abstracción

- Encapsulación: Debería ser posible ocultar lo interno de una parte.
- Composicionalidad: Debería ser posible combinar partes para construir nuevas partes.
- Instanciación/invocación: Debería ser posible crear muchas instancias de una parte basados en una sola definición.
- Extensionalidad: Debería ser posible extender una parte sin afectar a otras partes que la usen.

Conceptos para soportar esas propiedades

- El alcance léxico soporta la encapsulación.
- La programación de alto orden soporta la instanciación y la composicionalidad.
- El estado explícito soporta la extensionalidad.

Cuidados a tener con el estado explícito

- Con estado, aparecen los efectos de borde globales, que dificultan razonar sobre los programas.
- Con la encapsulación, los sistemas con estado pueden ser diseñados de manera que una propiedad bien definida, denominada un invariante, sea siempre cierta cuando se mira desde el exterior.
- En la construcción de sistemas complejos, el estado debe estar encapsulado de manera modular.

Diseño de sistemas y estado (2)

Propiedades para soportar el principio de abstracción

- Encapsulación: Debería ser posible ocultar lo interno de una parte.
- Composicionalidad: Debería ser posible combinar partes para construir nuevas partes.
- Instanciación/invocación: Debería ser posible crear muchas instancias de una parte basados en una sola definición.
- Extensionalidad: Debería ser posible extender una parte sin afectar a otras partes que la usen.

Conceptos para soportar esas propiedades

- El alcance léxico soporta la encapsulación.
- La programación de alto orden soporta la instanciación y la composicionalidad.
- El estado explícito soporta la extensionalidad.

Cuidados a tener con el estado explícito

- Con estado, aparecen los efectos de borde globales, que dificultan razonar sobre los programas.
- Con la encapsulación, los sistemas con estado pueden ser diseñados de manera que una propiedad bien definida, denominada un invariante, sea siempre cierta cuando se mira desde el exterior.
- En la construcción de sistemas complejos, el estado debe estar concentrado en unos pocos componentes.

Diseño de sistemas y estado (2)

Propiedades para soportar el principio de abstracción

- Encapsulación: Debería ser posible ocultar lo interno de una parte.
- Composicionalidad: Debería ser posible combinar partes para construir nuevas partes.
- Instanciación/invocación: Debería ser posible crear muchas instancias de una parte basados en una sola definición.
- Extensionalidad: Debería ser posible extender una parte sin afectar a otras partes que la usen.

Conceptos para soportar esas propiedades

- El alcance léxico soporta la encapsulación.
- La programación de alto orden soporta la instanciación y la composicionalidad.
- El estado explícito soporta la extensionalidad.

Cuidados a tener con el estado explícito

- Con estado, aparecen los efectos de borde globales, que dificultan razonar sobre los programas.
- Con la encapsulación, los sistemas con estado pueden ser diseñados de manera que una propiedad bien definida, denominada un invariante, sea siempre cierta cuando se mira desde el exterior.
- En la construcción de sistemas complejos, el estado debe estar concentrado en unos pocos componentes.

Diseño de sistemas y estado (2)

Propiedades para soportar el principio de abstracción

- Encapsulación: Debería ser posible ocultar lo interno de una parte.
- Composicionalidad: Debería ser posible combinar partes para construir nuevas partes.
- Instanciación/invocación: Debería ser posible crear muchas instancias de una parte basados en una sola definición.
- Extensionalidad: Debería ser posible extender una parte sin afectar a otras partes que la usen.

Conceptos para soportar esas propiedades

- El alcance léxico soporta la encapsulación.
- La programación de alto orden soporta la instanciación y la composicionalidad.
- El estado explícito soporta la extensionalidad.

Cuidados a tener con el estado explícito

- Con estado, aparecen los efectos de borde globales, que dificultan razonar sobre los programas.
- Con la encapsulación, los sistemas con estado pueden ser diseñados de manera que una propiedad bien definida, denominada un invariante, sea siempre cierta cuando se mira desde el exterior.
- En la construcción de sistemas complejos, el estado debe estar concentrado en unos pocos componentes.