Objetos con buen comportamiento

Conceptos principales

- Prueba
- Depuración
- Prueba de unidad
- Prueba automatizada
- Especificación para mantenimiento

Se debe tratar con errores

- Los errores iniciales son usualmente *errores sintácticos*.
  - El compilador los señalará.
- Los errores posteriores son usualmente *errores de lógica*.
  - El compilador no es de ayuda.
  - Se los conoce también como *bugs*.
- Algunos errores de lógica no tienen una manifestación obvia immediata.
  - El software comercial raramente está libre de errores.

Prevención vs. Detección (Desarrollador vs. Mantenedor)

- Se puede disminuir la probabilidad de cometer errores.
  - Usando técnicas de ingeniería de software, como la encapsulación.
- Se pueden mejorar las chances de detección.
  - Usando prácticas de ingeniería de software, como la modularización y la documentación.
- Se pueden desarrollar habilidades de detección.

Prueba y depuración

- Ambas habilidades son cruciales.
- Las pruebas buscan la presencia de errores.
- La depuración busca la fuente de los errores.
  - La manifestación de un error puede ocurrir a alguna ‘distancia’ de su fuente.

Técnicas de prueba y depuración

- Pruebas de unidad (en BlueJ)
- Pruebas automatizadas.
- Seguimientos manuales.
- Instrucciones de impresión.
- Depuradores.

Pruebas de unidad (Unit testing)

- Cada unidad de una aplicación puede ponerse a prueba.
  - Métodos, clases, módulos (*package* en Java).
- Puede (debería) hacerse durante el desarrollo.
  - Encontrar y corregir errores temprano disminuye los costos de desarrollo (p. ej. tiempo de programación).
  - BlueJ provee herramientas de prueba.

Fundamentos de la prueba

- Entender qué debería hacer la unidad es conocer su *contrato*.
  - Se deben buscar violaciones al mismo.
  - Usar pruebas positivas y negativas.
- Probar los *límites o contorno*.
  - Cero, uno, lleno.
    - Buscar en una colección vacía.
    - Añadir a una colección llena.

Pruebas de unidad en BlueJ

- Se pueden crear objetos de clases individuales.
- Se pueden invocar métodos individuales.
- Los *inspectores* proveen una visión actualizada del estado de un objeto.
- Explorar por medio del proyecto ***agenda-diaria-prueba***.

Ej.: 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9

Pruebas automatizadas

- La buenas pruebas son un proceso creativo, pero ...
- ... las pruebas exhaustivas consumen tiempo y son repetitivas.
- Las pruebas de *regresión* involucran la re-ejecución de las pruebas.
- El uso de un *esquema* o *sistema de pruebas* puede disminuir algo de la complejidad.
  - Se escriben las clases para realizar la prueba.
  - La creatividad se enfoca en crearlas.

Ej.: 6.10, 6.11, 6.12, 6.13

Pruebas automatizadas

- Explorar por medio del proyecto

  agenda-diaria-prueba

  - Todavía es necesario el análisis humano de los resultados
- Explorar la automatización completa por medio de los proyectos

  agenda-diaria-prueba-junit-v1/2.

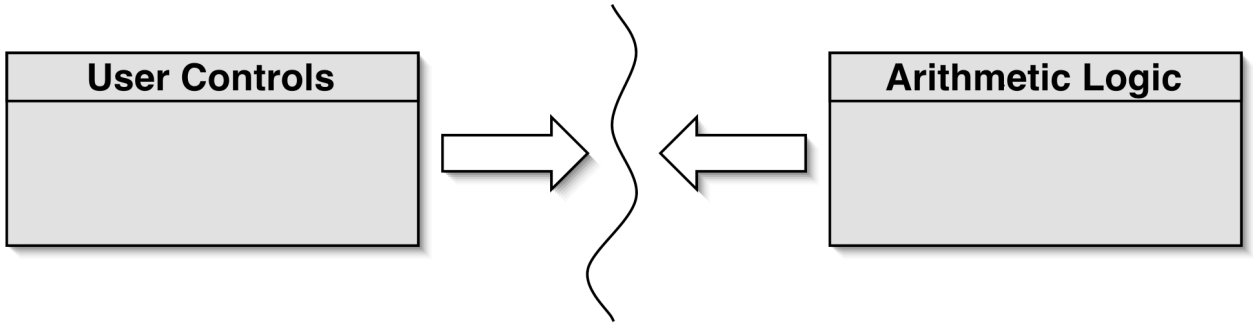
  - Solo se requiere la intervención si se reporta una falla. (www.junit.org)

Ej.: 6.14, 6.15, 6.16, 6.17, 6.18, 6.19, 6.20

Modularización e interfaces

- A menudo, las aplicaciones se componen con diferentes módulos.
  - De tal manera que diferentes equipos de personas puedan trabajar en ellos.
- La *interfaz* entre los módulos debe estar claramente *especificada*.
  - Soporta el desarrollo concurrente en forma independiente.
  - Incrementa la probabilidad de una integración exitosa.

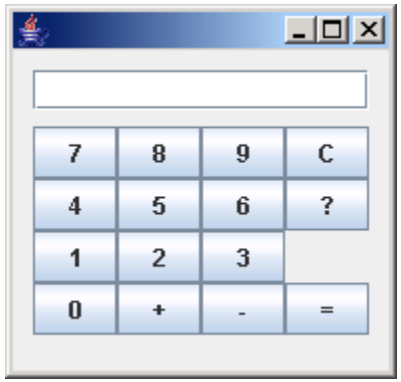
La modularización en una calculadora



- Cada módulo no necesita conocer los detalles de la implementación del otro.
  - Los controles de usuario podrían ser una IGU o un dispositivo de hardware.
  - La lógica podría ser hardware o software.

La signatura de los métodos como una interfaz

```
// Devuelve el valor que se mostrará.
public int getValorEnVisor();

// Se llama cuando se presiona un boton
public void numeroPresionado(int number);

// Se llama cuando se presiona (+).
public void mas();

// Se llama cuando se presiona (-).
public void menos();

// Se llama para completar un cálculo.
public void igual();

// Se llama para reiniciar.
public void limpiar();
```



Depuración

- Es importante desarrollar **habilidades** de lectura de código.
  - A menudo, la depuración se realizará sobre el código creado por otro.
- Existen técnicas y herramientas para suportar el proceso de depuración.
- Explorar por medio del proyecto ***calculadora-motor***.

Ej.: 6.21, 6.22, 6.23

Seguimiento manual

- Relativamente poco usado.
  - Un enfoque de baja tecnología.
  - Más poderoso que apreciado.
- ¡Aléjese de la computadora!
- ‘Corra’ un programa a mano.
- Vistas del depurador:
  - alto-nivel (*Step*)
  - bajo-nivel (*Step-into*)

Tabulación del estado de un objeto

- Usualmente, el comportamiento de un objeto se determina por su estado.
- El comportamiento incorrecto, es en general, el resultado de un estado incorrecto.
- Tabular los valores de todos los campos.
- Documentar los cambios de estado después de cada llamada a un método.

Ej.: 6.24

Seguimiento verbal

- Explicar a alguien qué es lo que está haciendo el código.
  - El oyente puede detectar el error.
  - El proceso de explicación puede ayudar a que uno mismo detecte el error.
- Existen procesos basados en grupos para realizar seguimientos formales o *inspecciones*.

Ej.: 6.25, 6.26, 6.27, 6.28, 6.29

Sentencias de impresión

- Es la técnica más popular.
- No se requieren herramientas especiales.
- Todos los lenguajes de programación la soportan.
- Es efectiva solo si los métodos de nivel superior están documentados.
- ¡La salida puede ser voluminosa!
- Su activación y desactivación requiere atención posterior.

Ej.: 6.30, 6.31, 6.32, 6.33

Elección de una estrategia de prueba

- Tome conciencia de las diferentes estrategias.
- Elija estrategias apropiadas para el punto del desarrollo.
- Automatícelas cuando sea posible.
  - Reduce el tedio.
  - Reduce el error humano.
  - Hace más probable las (re)pruebas.

Depuradores

- Los depuradores son específicos de lenguajes y ambientes.
  - BlueJ tene un depurador integrado.
- Soporta puntos de interrupción.
- Ejecución controlada mediante *Step* y *Step-into*.
- Secuencia de llamadas (pila o *stack*).
- Estado de un objeto.

Ej.: 6.35

Repaso

- Los errores son un factor en la vida de los programas.
- Las buenas técnicas de la ingeniería de software pueden reducir su ocurrencia.
- Las habilidades de hacer pruebas y depurar son senciales.
- Haga de las pruebas un hábito.
- Automatice las pruebas cuando sea posible.
- Practique un rango de habilidades de depuración..