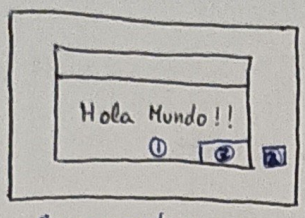
Persistencia

Class X {

X variable = new X ();



Componentes

↓

Forma

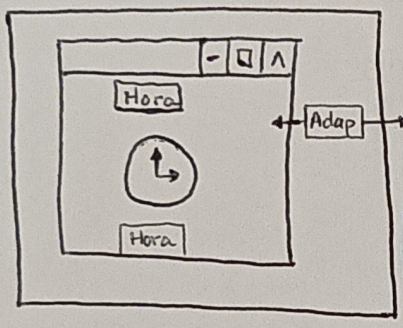
↙ Test/ Label ↘ Button

- Todos los objetos tienen todos los eventos

① Ventana { - visible, - tamaño, - posesión } { - tipos, - diálogos, - componentes, - portales }

② ratón { - icono, - color, - x, y, - dpi } { - tamaño, - pulsador 1, - pulsador 2, - pulsador 3 }

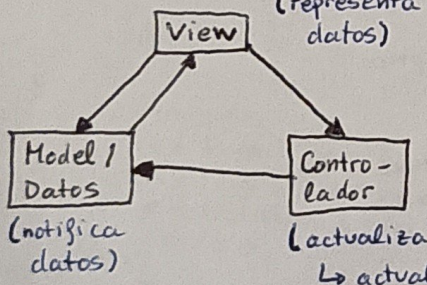
③ On Click ()



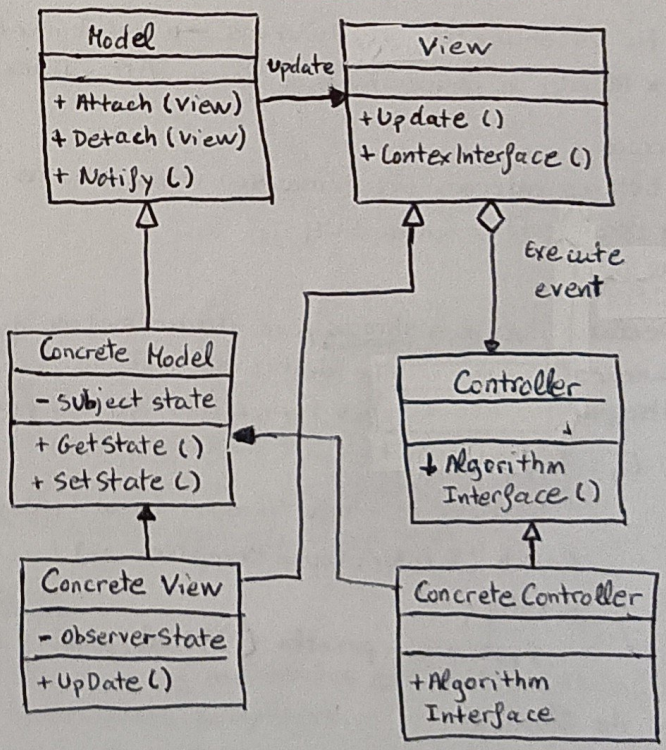
Usamos adaptadores entre ventanas

```
Class reloj {
  Datetrue d;
  String s;
  void setD (Datetime hora) {
    this.d = hora; }
  T.setText (hora) { };
  C.setTime (hora) { }; }
```

MVC



MVC – UML Complejo



Excepciones

- Importancia:
  - Son elementos anormales que alteran flujo normal programa
  - Si no se manejan, pueden dejar conexiones y recursos en estados inconsistentes
    - * Ej: Un error de entrada/salida puede dejar un archivo abierto
- Ventajas:
  - Declaración excepciones: los métodos declaran las excepciones
    - * Ej: Scanner (File Source) lanza FileNotFoundException si no se encuentra archivo
  - Obligación manejarlas: se necesita buena compilación para manejarlas
    - * Ej: El bloque try-catch atrapa FileNotFoundException
  - Separa lógica y manejo excepciones: mejora legibilidad código
    - * Ej: Try contiene lógica principal y catch gestiona errores
- Tipos:
  - Excepciones verificadas (checked):
    - Deben ser capturadas o declaradas → IOException
      - * Scanner (File Source) requiere manejo explícito de FileNotFoundException
  - Excepciones no verificadas (unchecked):
    - No es obligatorio capturarlas → NullPointerException
      - * Acceder a índice inválido lanza ArrayIndexOutOfBoundsException
  - Errores:
    - Errores internos programa son irrecuperables → VirtualMachineError
      - * Error crítico en la JVM
- Operaciones:
  - Declarar: usamos throws para firmar método que indica excepciones para lanzar
  - Lanzar: usamos throw dentro del método para lanzar excepción
  - Atrapar: utiliza bloques try-catch-finally para manejar excepciones
    - * Ej:

```
try{
    Scanner in = new Scanner (new File ("text.in"));}
Catch (FileNotFoundException ex){ ex.printStackTrace();}
Finally{
    System.out.println ("Finalizando...");}
```

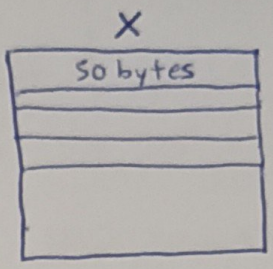
- Pilas de llamada:
  - Las excepciones se propagan en una pila de llamadas hasta encontrar un manejador apropiado.
  - Si no hay manejador, el programa termina → división por 0 lanza ArithmeticException
- Creación excepciones:
  - Podemos crear nuestras propias excepciones con Exception
    - * Ej:

```
public class MiException extends Exception{
    public MiException (String mensaje){
        super(mensaje);}
}
```

Persistencia

```
Class X {
  Atributos { String Nombre
  Métodos { void dimeNombre (){
            }
}
```

```
X variable = new X();
variable.Nombre = "Antonio";
manejador.guardar (Variable)
```



Lenguajes
- XML
- HTML
- MD

F datos
- Binarios
  - Enteros
  - Registros (R-Record)
- Texto
  - TXT
  - CSV
  - Ed.
- Interoperabilidad
  - JSON → Internet
  - XML → SOA
  - YAML → Contenedores / Infraestructuras

JSON

```
{ Nombre: Antonio,
  Edad: 48
  C: [ { x=0, y=1, radio: 7 }
       { x=1, y=3, radio: 9 } ] }
```

XML

```
Class Circulo {
  int x
  int y
  int radio }
```

```
Class X {
  Atributos { String nombre
              int Edad
              Circulo [] C
  Métodos { void dimeNombre
}
```

```
<!-- null />
<x>
  <nombre type: "String"> Antonio </nombre>
  <edad type: "int"> 48 </edad>
  <c type: "Array" num = 2>
    <circulo>
      <x type = "int"> 0 </x>
      <y type = "int"> 1 </y>
      <radio type = "int"> 7 </radio>
    </c>
</x>
```

```
<!-- XML >
<x nombre = "Antonio" Edad = 48>
  <c type = "Array" num = 2>
    <circulo x=0 y=1 radio = 7>
    <circulo x=1 y=3 radio = 9>
  </c>
</x>
```

```
Tipo: X
  nombre: Antonio
  edad: 48
  C:
    - x: 0
      y: 1
      radio: 7
    - x: 1
      y: 3
      radio: 9
```

JSON
- Necesita formato ligero y rápido
- Datos simples y estructura conocida
- Usado en aplicaciones web

XML
- Validar datos
- Requiere de metadatos con datos principales
- Documentos complejos

Seriación: convierte objeto en java en un flujo de bytes para guardarlo en un archivo.
Usa interfaz Serializable