Repaso

▼ Casting

- Implícito: el tipo de dato de origen es menor que el tipo de dato de destino. La conversion de realiza automáticamente por el compilador.
- Explícito: el tipo de dato de origen es mayor que el tipo de dato de destino. Para que se efectué el casting se debe escribir el tipo de datos de destino.
- Boolean es el único tipo de dato al que no se le puede hacer casting.
- Wrapper (parse) es casting explícito.
- `Instanceof` es un operador que sirve para conocer el tipo de objeto.

▼ Clases

- Los atributos de una clase son atributos de clase si se declaran como `static`.
- El modificador `final` se puede aplicar a clases, métodos y atributos.

▼ Herencia

- Una clase hija puede acceder directamente a los atributos del padre, si estos atributos se declaran como `protected`.
- La herencia entre clases es simple.

Herencia de clases abstractas

- Clases que no se pueden instanciar para que otras clases hereden de ella.
- Normalmente son la raíz de la jerarquía de clases y tienen definido el comportamiento de las subclases.
- Las clases derivadas de la clase abstracta son las que contienen la implementación y deben hacerlo con todos los métodos abstractos heredados.
- Características:
    - Pueden contener cero o más métodos abstractos.

    - Pueden contener métodos no abstractos.
    - Pueden contener atributos.
    - Pueden contener constructores, aunque no se instancien.
    - Si una clase derivada no implementa algún método abstracto heredado, se convierte en una clase abstracta y debe ser declarada como tal.

▼ Polimorfismo

- Polimorfismo estático o de sobrecarga: el overloading permite que una clase tenga varios métodos con el mismo nombre, pero con diferente número de parámetros.
- Polimorfismo dinámico: el overriding permite redefinir métodos heredados de la superclase en la subclase.
    - Sustituyen a la implementación original de la superclase.
    - Hay que añadir la etiqueta `@Override`

▼ Interfaces

Propiedades de las interfaces

- Una interface es una clase abstracta pura, sólo contiene la signatura de sus métodos.
- Puede contener la definición de datos, pero serán constantes.
- Puede declarar e implementar métodos por defecto, su código se ejecutará si una clase derivada no redefine el método.
- Puede implementar métodos estáticos que son propiedad de la clase y se declaran como static.
- Las características de la interface son las siguientes:
    - Todos los métodos son abstractos.
    - El acceso a una interface es público:
        - Métodos → `public`.
        - Atributos → `public static final`.
    - Las declaraciones de acceso son implícitas (no se necesita indicar el tipo de acceso).

- Son elementos de diseño.
- No puede contener constructores.

Herencia de interfaces

- Una clase que hereda de una interface debe implementar sus métodos.
- La palabra clave para realizar la herencia es `implements`.
- Una clase puede implementar múltiples interfaces (herencia múltiple).
- Una interfaz puede heredar múltiples interfaces. La palabra clave es `extends`.

Proliferación de nombre

- Si dos o más interfaces comparten la misma firma de método se producen errores de compilación.
- La solución a este problema implica llamar al método del interface deseado utilizando la palabra clave super. `interface.super.metodo()`.

▼ Streams y Buffer

- Los `Streams` se utilizan para vincular variables a la lectura o escritura de bytes con el periférico.
- Un `Stream` es un flujo de entrada, salida, entrada/salida y transversal.
- Para comunicarnos con el `Stream` utilizando cadenas de caracteres en lugar de bytes, asociamos a cada `Stream` un `Buffer` para evitar trabajar con bytes.

▼ Rutas, separadores y saltos de línea

- Las rutas relativas indican la ubicación de archivos especificando la ubicación del archivo respecto al programa sin necesidad de conocer la ruta raíz del sistema de archivos.
- `file.separator` → Devuelve el carácter que utiliza el sistema operativo para separar las carpetas de la ruta.
- `line.separator` → Devuelve el caracter que utiliza el sistema operativo para hacer un salto de línea.

▼ Serialización

- La serialización permite transformar un objeto en una secuencia de bytes para ser guardado en un fichero o enviado por la red, pudiendo posteriormente recomponerlo en un objeto serializado sin problemas.
- La clase a serializar tiene que implementar la interfaz `Serializable`. Esta interfaz no define ningún método, por lo que no tendremos que implementar nada a la clase.
- Clase `ObjectOutputStream` → Guarda objetos en un fichero.
- Clase `ObjectInputStream` → Recupera los datos del fichero.

Modificador `transient`

- Permite no guardar el valor de un atributo al serializar un objeto.
- Ejemplo: `private transient String password "qwerty123";`

▼ Excepciones

Capturas de excepciones

- Las excepciones se capturan y manejan con los bloques `try/catch`.
- Es obligatorio tratar las excepciones que generan una instrucción.
- Si salta una excepción del tipo especificado en el bloque `try`, se ejecuta el bloque `catch` correspondiente.
- El bloque de `Exception` es el último y sirve para capturar excepciones generales.
- El bloque `finally` se ejecuta siempre.

Tipos de excepciones

- Excepciones de Java → Excepciones del lenguaje.
- Excepciones de usuario → Excepciones creadas por el desarrollador para manejar los errores.
  - Hereda de la clase `Exception` y redefine sus métodos.

▼ Métodos `equals` y `hashcode`

- `equals()` → Compara que dos elementos sean iguales, pudiendo personalizarse para considerar criterios específicos.

- `hashCode()` → Compara que dos elementos en base a sus códigos hash. Es más rápida.
- Ambos métodos deben implementarse en clases personalizadas para definir la igualdad entre objetos.
- En estructuras hash como `HashMap`, primero llaman a `hashCode()` y luego a `equals()`. Si los códigos hash son diferentes, los objetos son distintos, pero si son iguales, se ejecuta `equals()` para una comparación detallada.

▼ Interfaces de usuario

Elementos

- `JButton` → Botón.
- `JLabel` → Etiqueta de texto.
- `JTextField` → Cuadrado de texto.
- `JCheckBox` → Casilla de verificación.
- `JRadioButton` → Botón de opciones.
- `JComboBox` → Lista desplegable.

Contenedores

- Son componentes especiales que agrupan otros componentes en una interfaz gráfica.
- Tipos de contenedores:
  - Contenedores de nivel superior:
    - `JFrame` → Ventana principal de la aplicación.
    - `JDialog` → Ventana de tipo diálogo.
    - `JApplet` → Zona dentro de un Applet para componentes Swing.
  - Contenedores de nivel inferior:
    - `JPanel`, `JScrollPane` y `JRootPane`.

Operaciones importantes

- `setDefaultCloseOperation(int)` → Define la acción al cerrar la ventana.
- `setVisible(boolean)` → Muestra la ventana.

- `setSize(int, int)` → Establece el tamaño.
- `setLocationRelativeTo(Component)` → Ubica la ventana respecto a otro componente.

Layout

- Esquemas de diseño para organizar componentes dentro de un contenedor.
- Tipos de layouts:
  - `FlowLayout` → Disposición de izquierda a derecha y de arriba abajo.
  - `BorderLayout` → Disposición en el centro o bordes del contenedor.
  - `GridLayout` → Cuadrícula.
  - `GridBagLayout` → Cuadrícula flexible.
  - `BoxLayout` → Disposición horizontal o vertical.
  - `SpringLayout` → Disposición basada en restricciones.
  - `NULL`

Eventos

- Acciones realizadas por el usuario en la interfaz gráfica.

Controladores de eventos

- Los controladores de eventos permiten ejecutar acciones específicas después de un evento.
- Java proporciona controladores de eventos (`listeners`) para manejar los eventos.
- Controladores de eventos (`listeners`) en Java:
  - `ActionListener` → Acción típica de un componente.
  - `FocusListener` → Acciones al obtener o perder el foco.
  - `KeyListener` → Respuesta a la pulsación de teclas.
  - `ItemListener` → Selección o deselección de una opción.
  - `MouseListener` → Respuesta al clic del ratón.

    - `MouseMotionListener` → Respuesta a acciones como arrastrar un elemento.
    - `WindowListener` → Respuesta a acciones sobre una ventana.
- Controlador puede ser anónimo o no.
- Es posible asignar múltiples controladores de eventos a un mismo componente.

▼ Procesos

Conceptos básicos

- Sistema operativo → Conjunto de programas que gestiona el hardware del ordenador y hace de intermediario entre las aplicaciones y los usuarios.
- Programa → Conjunto de instrucciones de código escritas en un lenguaje de programación para solucionar una tarea o necesidad.
- Proceso → Instancia de un programa en ejecución. Necesita otros recursos como el contador de instrucciones, el contenido de los registros y un espacio de memoria.
- Ejecutable → Fichero que contiene instrucciones en un determinado código y permite crear el proceso asociado a ese programa.
- Demonio → Proceso no interactivo controlado por el sistema operativo que se ejecuta en segundo plano y no dispone de una interfaz directa con el usuario.
- Sistema monoprocesador y multiprocesador.

Scheduler

- Encargado de decidir qué proceso entra en el procesador.
- La planificación tiene los niveles medio, corto y largo plazo.
- Los estados de un proceso son: Nuevo → Listo → Ejecución → Bloqueado → Terminado.

Algoritmos de planificación

- **FIFO (First Input First Output)**: Los procesos se ejecutan en orden de llegada y los siguientes deberán esperar su turno.

- **SJF (Short Job First)**: Se ejecuta primero el más corto (rápido).
- **SRTF (Short Remaining Time First)**: De los procesos en espera, selecciona el menos tiempo le queda.
- **Algoritmo por prioridades**: Asigna una prioridad a cada proceso al entrar en la CPU. Se ejecutaran los de mayor prioridad.
- **RR (Round Robin)**: Se asigna un tiempo (quantum) a cada proceso, tras el cual abandona la CPU y da paso al siguiente proceso.