# UT01. DESARROLLO DEL SOFTWARE.

Entornos de Desarrollo

1DAW – C.I.F.P. Carlos III - Cartagena

# 4. Fases del desarrollo de una aplicación

# 2. Diseño

- Diseño orientado a objetos-DOO
  - Se parte de un análisis orientado a objetos-AOO
    - Se definen las clases, operaciones, atributos, relaciones, comportamientos y comunicaciones.
    - Cuatro capas de DOO:
      - → **Subsistema**. Diseño de los subsistemas que implementan las funciones principales del sistema
      - → **Clases y objetos**. Especifica la arquitectura de objetos global y la jerarquía de clases requerida para implementar un sistema.
      - → **Mensajes**. Indica cómo se realiza la colaboración entre objetos
      - → **Responsabilidades**. Identifica las operaciones y atributos que caracterizan cada clase.
  - **UML**-*Unified Modeling Language*. Estándar de la mayor parte de las metodologías de desarrollo orientado a objetos.
    - Lenguaje de modelado basado en diagramas para expresar la realidad donde se ignoran los detalles de menor importancia

# 3. Codificación

- Transforma las especificaciones del diseño en un conjunto de instrucciones escritas en un lenguaje de programación, almacenadas dentro de un programa.
- Existen unas reglas o normas de escritura de código fuente.
- Esta tarea la realiza el programador y tiene que cumplir exhaustivamente con todos los datos impuestos en el análisis y en el diseño de la aplicación.
- Estados:
  - **Código Fuente**: escrito por los programadores usando algún lenguaje de programación de alto nivel.
  - **Código Objeto**: código binario resultado de compilar el código fuente. Compilación → traducción de una sola vez del programa (compilador). Interpretación → traducción y ejecución simultánea del programa línea a línea.
  - **Código Ejecutable**: código binario resultante de enlazar los archivos de código objeto con rutinas y bibliotecas. Es el código máquina directamente inteligible por la computadora.

# 3. Codificación

- **Fuente**
  - Código de instrucciones de alto nivel ➔ traducción al lenguaje máquina.
- Previamente se diseña un algoritmo en pseudocódigo
  - Se debe partir de las etapas anteriores de análisis y diseño.
  - Se diseñará un algoritmo que simbolice los pasos a seguir para la resolución del problema.
  - Se elegirá un Lenguajes de Programación de alto nivel apropiado para las características del software que se quiere codificar.
  - Se procederá a la codificación del algoritmo antes diseñado.

# 3. Codificación

- Un aspecto importante: su licencia.
  - **Código fuente abierto**. Disponible para que cualquier usuario pueda estudiarlo, modificarlo o reutilizarlo.
  - **Código fuente cerrado**. No tenemos permiso para editarlo.

# 3. Codificación

- **Objeto.**
  - Código intermedio
  - Resultado del proceso de traducción pero todavía no ejecutable.
  - Código binario distribuido en varios archivos libre de errores sintácticos y semánticos.
  - Compilación vs. interpretación

# 4. Pruebas

- Imprescindible para asegurar:
  - **Verificación,** se refiere al conjunto de actividades que tratan de comprobar si se está construyendo el producto correctamente *-el software cumple los requisitos especificados-* y
  - **Validación,** se refiere al conjunto de actividades que tratan de comprobar si el producto es correcto *-hace lo que el usuario desea-*

del software construido.

- Tipos de pruebas:
  - **Unitarias**
    - Probar las diferentes partes de software y comprobar su funcionamiento (por separado, de manera independiente).
    - **JUnit** es el entorno de pruebas para Java.
  - **Integración**
    - Comprueba el funcionamiento del sistema completo: con todas sus partes interrelacionadas, después de las unitarias.
- **Beta Test.**
  - Prueba final que se realiza sobre el entorno de producción donde el software va a ser utilizado por el cliente.

# 5. Documentación

- Todas las etapas en el desarrollo de software deben quedar perfectamente documentadas.
- Uso:
  - Dar toda la información a los usuarios de nuestro software
  - Poder acometer futuras revisiones del proyecto
- Tres grandes documentos en el desarrollo de software
  - Guía técnica
  - Guía de uso
  - Guía de instalación

# 5. Documentación

- **Guía técnica**
  - Para un correcto desarrollo y permitir mantenimiento
  - Uso por parte de los analistas y programadores.
  - Contiene el diseño, codificación y pruebas realizadas
- **Guía de uso**
  - Da a los usuarios finales información para su uso.
  - Uso por parte de los clientes (usuarios finales)
  - Contiene una descripción de la aplicación, ejecución, ejemplos de uso, requisitos del software, y solución de posibles problemas.
- **Guía de instalación**
  - Para garantizar la correcta instalación de la aplicación.
  - Personal informático responsable con los usuarios finales.
  - Contiene puesta en marcha, explotación y seguridad del sistema.

# 6. Explotación

- La explotación es la fase en que los usuarios finales conocen la aplicación y comienzan a utilizarla.
- Después de todas las fases anteriores, sin errores y documentación.
- **Instalación:**
  - Transferencia de los programas al computador del usuario cliente, configuración y verificación.
  - Presencia del cliente
  - Beta Test
- **Configuración.**
  - Asignación de los parámetros de funcionamiento normal de la empresa.
  - Prueba de la aplicación.
  - Posibilidad de la realización por parte del cliente con guía de instalación.
  - Posibilidad de programar la configuración de manera que se realice automáticamente tras instalarla.
- **Producción normal**
  - Explotación por parte del cliente

# 7. Mantenimiento

- La etapa de mantenimiento es la más larga de todo el ciclo de vida del software.
- Los tipos de cambios del mantenimiento del software :
  - **Perfectivos**: Para mejorar la funcionalidad del software.
  - **Evolutivos**: El cliente tendrá en el futuro nuevas necesidades. Por tanto, serán necesarias modificaciones, expansiones o eliminaciones de código.
  - **Adaptativos**: Modificaciones, actualizaciones... para adaptarse a las nuevas tendencias del mercado, a nuevos componentes hardware, etc.
  - **Correctivos**: La aplicación tendrá errores en el futuro (sería utópico pensar lo contrario).

# 4. Lenguajes de programación

# Características de los lenguajes de programación

- **Lenguaje máquina:**
  - Sus instrucciones son combinaciones de unos y ceros.
  - Es el único lenguaje que entiende directamente el ordenador. (No necesita traducción).
  - Fue el primer lenguaje utilizado.
  - Es único para cada procesador (no es portable de un equipo a otro).
  - Hoy día nadie programa en este lenguaje.
- **Lenguaje ensamblador:** https://es.wikipedia.org/wiki/Lenguaje_ensamblador
  - Sustituyó al lenguaje máquina para facilitar la labor de programación.
  - En lugar de unos y ceros se programa usando mnemotécnicos (instrucciones complejas).
  - Necesita traducción al lenguaje máquina para poder ejecutarse.
  - Sus instrucciones son sentencias que hacen referencia a la ubicación física de los archivos en el equipo.
  - Es difícil de utilizar.

# Características de los lenguajes de programación

- **Lenguaje de alto nivel basados en código:**
  - Sustituyeron al lenguaje ensamblador para facilitar más la labor de programación.
  - En lugar de mnemotécnicos, se utilizan sentencias y órdenes derivadas del idioma inglés. (Necesita traducción al lenguaje máquina).
  - Son más cercanos al razonamiento humano.
  - Son utilizados hoy día, aunque la tendencia es que cada vez menos.
- **Lenguajes visuales:**
  - Están sustituyendo a los lenguajes de alto nivel basados en código.
  - En lugar de sentencias escritas, se programa gráficamente usando el ratón y diseñando directamente la apariencia del software.
  - Su correspondiente código se genera automáticamente.
  - Necesitan traducción al lenguaje máquina.
  - Son completamente portables de un equipo a otro.

https://www.edix.com/es/instituto/lenguajes-de-programacion/

# Conceptos y características

- Un lenguaje de programación es el conjunto de:
  - **Alfabeto**: conjunto de símbolos permitidos.
  - **Sintaxis**: normas de construcción permitidas de los símbolos del lenguaje.
  - **Semántica**: significado de las construcciones para hacer acciones válidas.
- **Clasificación según la cercanía al lenguaje humano**
  - De alto nivel: más próximos al razonamiento humano.
  - De bajo nivel: más próximos al funcionamiento interno de la computadora (Ensamblador y Máquina).
- **Clasificación según la técnica de programación utilizada:**
  - Estructurados: Pascal, C, etc.
  - Orientados a Objetos: C++, Java, Ada, Delphi, etc.
  - Visuales: Visual Basic.Net, Borland Delphi, etc.

# Estructurados

- Uso de tres tipos de sentencias o estructuras de control:
  - Sentencias secuenciales.
  - Sentencias selectivas (condicionales).
  - Sentencias repetitivas (iteraciones o bucles).

| Ventajas | Inconvenientes |
| --- | --- |
| •Los programas son fáciles de leer, sencillos y rápidos.<br>•El mantenimiento de los programas es sencillo.<br>•La estructura del programa es sencilla y clara. | •Todo el programa se concentra en un único bloque.<br>•No permite reutilización eficaz de código. ➔ programación modular. |

# Orientados a objetos

- Conjunto de objetos **independientes y reutilizables** que colaboran entre ellos para realizar acciones.
- No es una programación tan intuitiva como la estructurada.
- El código es reutilizable.
- Facilidad para localizar y depurar un error en un objeto.
- Los **objetos** compuestos de atributos.
- **Clase** como una colección de objetos con características similares.
- **Métodos**: comunicación entre objetos.

# Fases en la obtención de código

- **Fuente**
  - Código de instrucciones de alto nivel ➔ traducción al lenguaje máquina.
  - Previamente se diseña un algoritmo en pseudocódigo
    - Se debe partir de las etapas anteriores de análisis y diseño.
    - Se diseñará un algoritmo que simbolice los pasos a seguir para la resolución del problema.
    - Se elegirá una Lenguajes de Programación de alto nivel apropiado para las características del software que se quiere codificar.
    - Se procederá a la codificación del algoritmo antes diseñado.

# Fases en la obtención de código

- Un aspecto importante: su licencia.
  - **Código fuente abierto**. Disponible para que cualquier usuario pueda estudiarlo, modificarlo o reutilizarlo.
  - **Código fuente cerrado**. No tenemos permiso para editarlo.

# Fases en la obtención de código

- **Objeto.**
  - Código intermedio
  - Resultado del proceso de traducción pero todavía no ejecutable.
  - Código binario distribuido en varios archivos libre de errores sintácticos y semánticos.
  - Compilación vs. interpretación

# Fases en la obtención de código

- **Ejecutable**
  - Un archivo resultado de enlazar los archivos de código objeto (linker)
  - Ejecutado y controlado por el sistema operativo.

# Máquinas virtuales

- Tipo especial de software cuya misión es separar el funcionamiento del ordenador de los componentes hardware instalados.
- Garantiza la portabilidad de las aplicaciones
- Características
  - Compilación de código fuente➔bytecode
  - M.Virtual verifica el bytecode.

# Framework

- Estructura de ayuda al programador para desarrollar proyectos sin partir desde cero.
- Plataforma software donde están definidos programas soporte, bibliotecas, lenguaje interpretado, etc., que ayuda a desarrollar y unir los diferentes módulos o partes de un proyecto.
- *Ventajas* de utilizar un framework:
  - **Desarrollo rápido** de software.
  - **Reutilización** de partes de código para otras aplicaciones.
  - **Diseño** uniforme del software.
  - **Portabilidad** de aplicaciones de un computador a otro, ya que los bytecodes que se generan a partir del lenguaje fuente podrán ser ejecutados sobre cualquier máquina virtual.

# Frameworks

- *Inconvenientes*:
  - Gran dependencia del código respecto al framework utilizado.
  - La instalación e implementación del framework en nuestro equipo consume bastantes recursos del sistema.
- Ejemplos de Frameworks:
  - .NET es un framework para desarrollar aplicaciones sobre Windows.
  - Spring de Java
  - Laravel
  - …
- https://kinsta.com/es/blog/frameworks-php/

# Entornos de ejecución

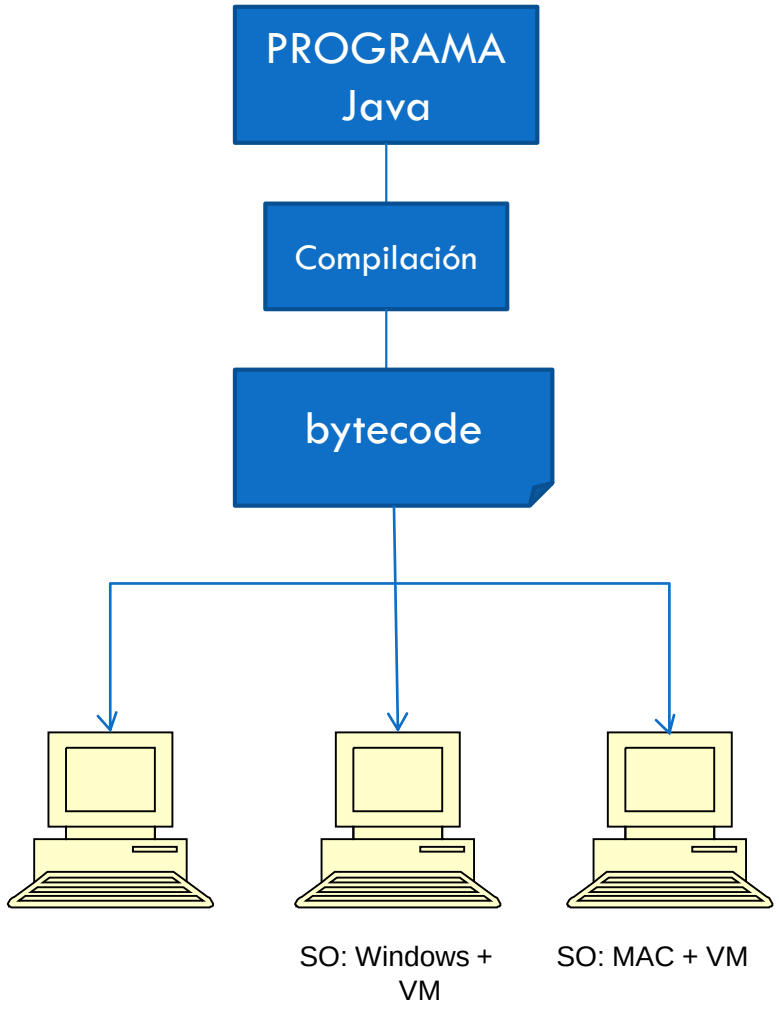
- Un entorno de ejecución es un servicio de máquina virtual que sirve como base software para la ejecución de programas.
- *Actividades* del entorno durante la ejecución:
  - Configurar la memoria principal
  - Enlazar los archivos del programa con las bibliotecas existentes y con los subprogramas creados.
  - Depurar los programas: comprobar la existencia (o no existencia) de errores semánticos del lenguaje (los sintácticos ya se detectaron en la compilación).

# Java runtime environment

- JRE -Java Runtime Environment.
- Conjunto de utilidades que permitirá la ejecución de programas java sobre cualquier tipo de plataforma.
- **Componentes**
  - JMV o JVM -Máquina virtual Java -programa que interpreta el código de la aplicación escrito en Java.
  - Bibliotecas de clase estándar que implementan el API de Java.
- Descargar el programa JRE.
- Java es software libre.

# Java runtime environment

- Lenguaje que genera Java = lenguaje intermedio interpretable por una máquina virtual instalada en el ordenador a ejecutar.
- Máquina virtual de Java = máquina ficticia que traduce las instrucciones máquina-ficticia en instrucciones para la máquina real.



# Java runtime environment

- Java es compilador e intérprete.
- Compilador: compila a bytecode
- Intérprete: ejecuta el código en la máquina real.

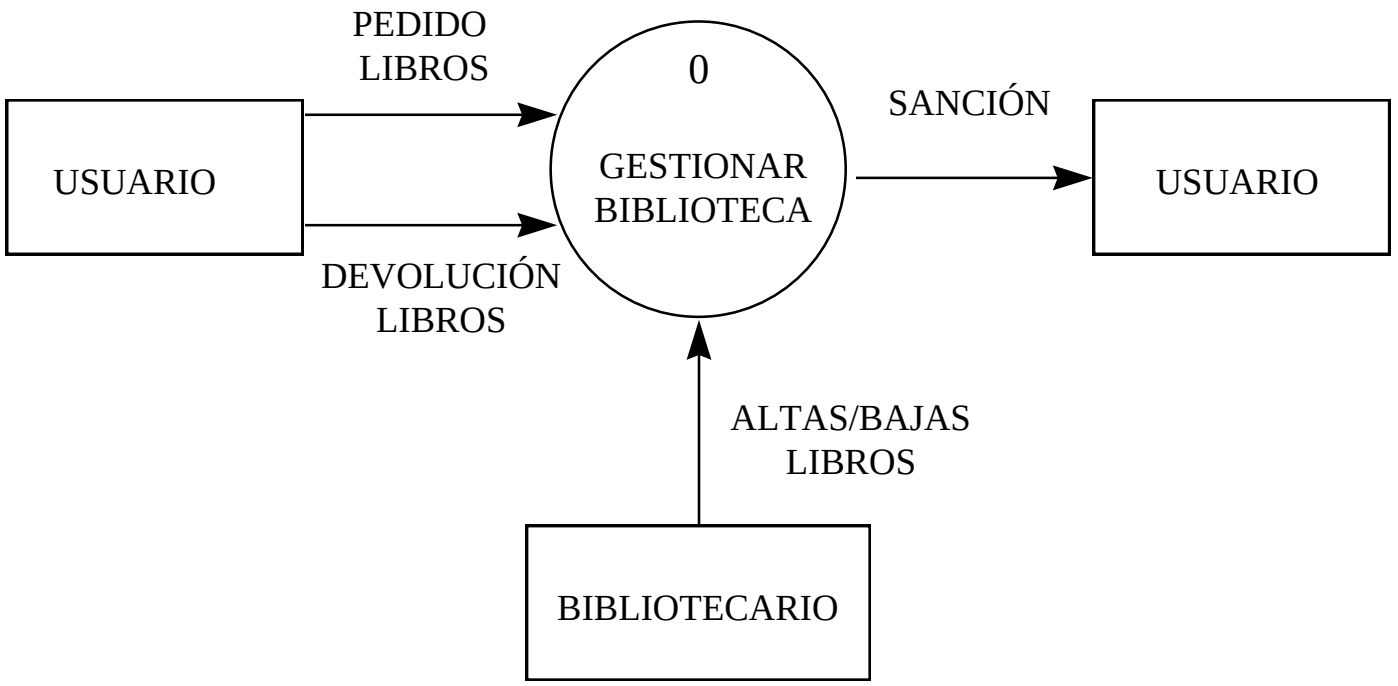
# Java runtime environment

- **JDK-**Java Development Kit.
  - No incluye herramienta gráfica
  - Incluye el **JRE**-Java Runtime Environment-: incluye como mínimo componentes para ejecutar una aplicación Java (máquina virtual y librerías de clase).

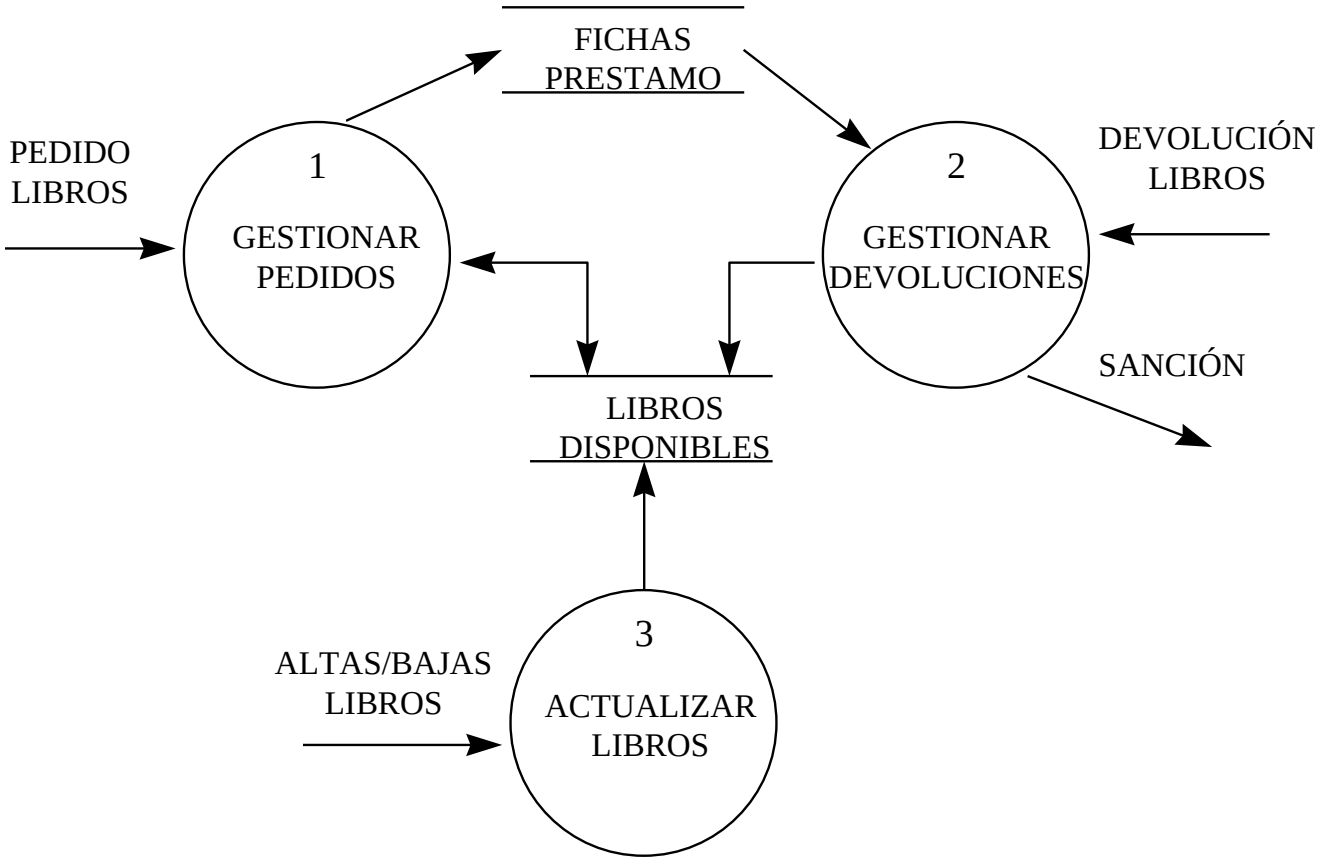
- Herramientas de consola de JDK:
  - **java**: VM de java
  - **javac**: compilador
  - **javap**: desensamblador de clases
  - **jdb**: depurador de consola
  - **javadoc**: generador de documentación
  - **Appletviewer**: visor de Applets.

# Anexo I.- Ejemplo de DFD

**DIAGRAMA DE CONTEXTO**



# DIAGRAMA 0: GESTIONAR BIBLIOTECA



# DIAGRAMA 2: GESTIONAR DEVOLUCIONES