# Curso de desarrollo de software

## Examen Final

### Reglas de la evaluación

- Queda terminantemente prohibido el uso de herramientas como ChatGPT, WhatsApp, o cualquier herramienta similar durante la realización de esta prueba. El uso de estas herramientas, por cualquier motivo, resultará en la anulación inmediata de la evaluación.
- Las respuestas deben presentarse con una explicación detallada, utilizando términos técnicos apropiados. La mera descripción sin el uso de terminología técnica, especialmente términos discutidos en clase, se considerará insuficiente y podrá resultar en que la respuesta sea marcada como incorrecta.
- Toda respuesta que incluya código debe ser acompañada de una explicación. La ausencia de esta explicación implicará que la pregunta se evaluará como cero.
- Utiliza imágenes para explicar o complementar las respuestas, siempre y cuando estas sean de creación propia y relevantes para la respuesta, como es el caso de la verificación de los pasos de pruebas.
- Cada estudiante debe presentar su propio trabajo. Los códigos iguales o muy parecidos entre sí serán considerados como una violación a la integridad académica, implicando una copia, y serán sancionados de acuerdo con las políticas de la universidad.
- La claridad, orden, y presentación general de las evaluaciones serán tomadas en cuenta en la calificación final. Se espera un nivel de profesionalismo en la documentación y presentación del código y las respuestas escritas.

### Instrucciones de entrega para la prueba calificada

Para asegurar una entrega adecuada y organizada de su prueba calificada, sigan cuidadosamente las siguientes instrucciones. El incumplimiento de estas pautas resultará en que su prueba no sea revisada, sin importar el escenario.

- Cada estudiante debe tener un repositorio personal en la plataforma designada para el curso (por ejemplo, GitHub, GitLab, etc.). El nombre del repositorio debe incluir su nombre completo o una identificación clara que permita reconocer al autor del trabajo fácilmente.
- Dentro de su repositorio personal, deben crear una carpeta titulada **ExamenFinal-CC3S2.** Esta carpeta albergará todas las soluciones de la prueba.
- Dentro de la carpeta **ExamenFinal-CC3S2,** deben crear dos subcarpetas adicionales, de acuerdo con el número de ejercicios dados en la prueba

- Todas las respuestas y soluciones deben ser documentadas, completas y presentadas en archivos Markdown (.md) dentro de las subcarpetas correspondientes a cada ejercicio. Los archivos Markdown permiten una presentación clara y estructurada del texto y el código.
- Cada archivo Markdown de respuesta debe incluir el nombre del autor como parte del nombre del archivo. Esto es para asegurar la correcta atribución y reconocimiento del trabajo individual.

- No se aceptarán entregas en formatos como documentos de Word (.docx) o archivos PDF. La entrega debe cumplir estrictamente con el formato Markdown (.md) como se especifica.
- Es crucial seguir todas las instrucciones proporcionadas para la entrega de su prueba calificada. Cualquier desviación de estas pautas resultará en que su prueba no sea considerada para revisión.
- Antes de la entrega final, verifique la estructura de su repositorio, el nombramiento de las carpetas y archivos, y asegúrese de que toda su documentación esté correctamente formateada y completa.

**Problema**

Estás encargado de desarrollar y probar un sistema que desencadena eventos avanzados basados en combinaciones complejas de condiciones booleanas externas relacionadas con el clima. Este sistema debe ser diseñado de manera limpia, siguiendo principios de diseño como responsabilidad única, código limpio y TDD.

El proyecto se desarrollará en dos sprints, donde se aplicarán técnicas de validación de pruebas, refactorización y contenerización utilizando Docker.

### Entradas y salidas del sistema

#### *Entrada*

El sistema recibe un conjunto de condiciones climáticas externas como entrada. Estas condiciones pueden incluir:

**Temperatura** (en grados Celsius)
**Cantidad de Lluvia** (en mm)
**Velocidad del Viento** (en km/h)
**Humedad** (en porcentaje)
**Presión Atmosférica** (en hPa)

Cada una de estas condiciones será evaluada para determinar si se cumplen ciertos criterios booleanos que desencadenan eventos específicos.

#### *Salida*

El sistema genera eventos basados en las combinaciones de las condiciones climáticas evaluadas. Las salidas pueden ser:

**Alertas** (Alerta de Lluvia Intensa, Alerta de Viento Fuerte)
**Acciones automáticas** (Activar Sistema de Riego, Cerrar Persianas)
**Notificaciones** (Enviar Notificación a Usuarios)

Las salidas deben estar claramente definidas y documentadas en función de las combinaciones específicas de condiciones climáticas.

## Primer Sprint: Estrategia de validación de pruebas, stubs y fakes (8 puntos)

### *Objetivos*

- **Desarrollo inicial del sistema**: Crear el sistema con un diseño limpio y clases con responsabilidad única.
- **Implementación de TDD**: Utilizar el desarrollo dirigido por pruebas (TDD) para asegurar que cada componente cumpla con su responsabilidad.
- **Estrategia de validación de pruebas**: Definir y aplicar una estrategia de validación que incluya stubs y fakes para simular las condiciones climáticas externas.
- **Refactorización y código limpio**: Asegurar que el código sea limpio y fácil de mantener mediante la refactorización continua.
- **Métricas de calidad**: Establecer métricas de calidad para evaluar la cobertura de pruebas y la complejidad del código.

### *Actividades*

1. **Diseño y implementación inicial**:
    - Diseñar las clases responsables de manejar las condiciones climáticas (Temperatura, Lluvia, Viento).
    - Implementar estas clases asegurando que cada una tenga una única responsabilidad.
2. **Desarrollo con TDD**:
    - Escribir pruebas unitarias para cada clase antes de implementar la funcionalidad.
    - Implementar la funcionalidad necesaria para que las pruebas pasen.
    - Refactorizar el código después de que las pruebas pasen para mantenerlo limpio.
3. **Validación de pruebas con stubs y fakes**:
    - Crear stubs para simular las condiciones climáticas
    - Implementar fakes para simular escenarios de prueba más complejos
4. **Refactorización y código limpio**:
    - Refactorizar el código regularmente para mejorar la legibilidad y mantenibilidad.
    - Aplicar principios de diseño limpio y patrones de diseño adecuados.
5. **Métricas de calidad**:
    - Utilizar herramientas para medir la cobertura de pruebas (Jacoco)
    - Evaluar la complejidad del código utilizando métricas como la complejidad ciclomática.

### *Entregables*

- Código fuente inicial del sistema con clases bien diseñadas.
- Conjunto de pruebas unitarias y de integración utilizando stubs y fakes.
- Métricas de cobertura de pruebas y complejidad del código.
-

## Segundo Sprint: Uso de docker y mejora de la estrategia de pruebas (12 puntos)

### *Objetivos*

- **Contenerización del sistema**: Utilizar Docker para contenerizar la aplicación y asegurar su portabilidad.
- **Refinamiento del tDD**: Continuar utilizando TDD para cualquier nueva funcionalidad o mejora.
- **Mejora de la estrategia de pruebas**: Integrar la estrategia de pruebas en un entorno Dockerizado.
- **Refactorización y código limpio**: Continuar refactorizando el código para mantener su calidad.
- **Métricas de calidad**: Continuar monitoreando y mejorando las métricas de calidad del código.

### *Actividades*

1. **Contenerización del Sistema**:
   - Crear un Dockerfile para construir la imagen de la aplicación.
   - Configurar un docker-compose.yml si se necesitan múltiples servicios (bases de datos, servicios de simulación de clima).
2. **Refinamiento del TDD**:
   - Escribir nuevas pruebas para cualquier funcionalidad adicional.
   - Asegurar que todas las pruebas existentes pasen en el entorno Dockerizado.
3. **Mejora de la estrategia de pruebas**:
   - Integrar las pruebas unitarias y de integración en el pipeline de Docker.
   - Asegurar que los stubs y fakes funcionen correctamente en el entorno contenerizado.
4. **Refactorización y código limpio**:
   - Continuar refactorizando el código para mejorar la calidad y mantener la adherencia a los principios de diseño limpio.
5. **Métricas de Calidad**:
   - Monitorear la cobertura de pruebas y la complejidad del código en el entorno Dockerizado.
   - Utilizar herramientas de análisis de código para asegurar la calidad.

### *Entregables*

- Imágenes Docker y archivos de configuración (Dockerfile, docker-compose.yml).
- Código fuente actualizado con nuevas funcionalidades y mejoras.
- Conjunto de pruebas actualizado y funcionando en el entorno Docker.
- Reportes de métricas de calidad del código.

### *Salida esperada*

Cuando ejecutas la aplicación con las condiciones climáticas especificadas (Temperatura de 35°C, Lluvia de 25mm, Viento de 55km/h), deberías ver la siguiente salida en la consola:

Alerta: Temperatura Alta!

Alerta: Lluvia Intensa!

Alerta: Viento Fuerte!