1. Permite ocultar unos miembros de la clase y visibilizar otros generando niveles de acceso.
2. La creación de estructuras representacionales de objetos o conceptos del mundo real para ser usados como tipos de datos definidos por el usuario.
3. Permite la reutilización de código y el uso del polimorfismo.
4. Hace posible el tratar a una clase heredada como si fuese uno de sus ancestros.
5. Hace referencia a una instancia de una clase.
6. Modelo o diseño de un tipo de dato que tiene atributos y comportamientos.
7. Nos permite resolver problemas creando modelos de la realidad y sus conceptos disminuyendo errores gracias a la reutilización de código.
8. Una forma o estilo difundido de programación que permite abordar cierta clase de problemas desde una perspectiva específica.
9. Atributos y métodos que forman parte de una clase.
10. Solamente pueden ser accedidos a lo interno de la propia clase.
11. Solamente pueden ser accedidos por clases del mismo paquete o que hereden de esta.
12. Pueden ser accedidos desde cualquier clase externa que herede o no, que esté o no dentro del mismo paquete.
13. Clases que se encuentran físicamente dentro de una misma carpeta.
14. Comportamientos de una clase.
15. Características de los objetos.
16. Permite usar tipos de datos de clases que no están en el mismo paquete.
17. Permite crear constantes.
18. Permite usar miembros de una clase sin declarar una instancia de esta.
19. Son la forma de asegurar la calidad a través del tiempo.
20. Librería que se usa para crear pruebas unitarias en Java.
21. Tipo específico de método de pruebas que se usa para comparar equidad.
22. Anotación que se usa para crear un método de prueba unitaria.
23. Nombre del paquete en el que deberíamos tener nuestras pruebas unitarias.
24. Tabulación de nuestro código que permite visualizar que bloque se encuentra dentro de que otro bloque y facilita la legibilidad y por ende el mantenimiento.
25. Método especial de una clase que permite inicializar sus atributos, crear instancias de la misma clase y que no devuelve nada
26. Método que permite asignar un valor a un miembro privado de una clase
27. Método que permite obtener el valor de un miembro privado de una clase
28. Palabra reservada que se usa para heredar
29. Palabra reservada que se usa para invocar el comportamiento del ancestro.
30. Palabra reservada para hacer referencia a los miembros de la propia clase.

No se ha presentado ningún contenido en esta tarea.

# Proyecto programado del curso

## Competencias claves a desarrollar por parte de los estudiantes con el desarrollo de este proyecto

1. El estudiante puede programar de pequeños sistemas utilizando el paradigma de programación orientada a objetos
2. El estudiante puede asegurar calidad a través de pruebas unitarias en Java utilizando JUnit
3. El estudiante puede diseñar sistemas pequeños orientados a objetos usando los diagramas: casos de uso, clases, estados y secuencia
4. El estudiante puede trabajar en equipos de desarrollo pequeños según el flujo de control de versiones básico de git

## Etapas del proyecto

1. Primer avance
2. Entrega final

### El primer avance deberá contener

1. El diseño de la solución usando los diagramas vistos en clase
2. La programación del código base del proyecto que corra en una aplicación de consola a modo de backend
3. Las pruebas unitarias a todos y cada uno de los métodos públicos de las clases del proyecto.
4. Un informe de trabajo grupal dónde quede en evidencia el rol de todos y cada uno de los miembros del equipo
5. Completar el formulario individual de autoevaluación.
6. Un informe de no más de media página por estudiante con la evaluación que hace cada integrante del trabajo de sus compañeros (Coevaluación)
7. Un informe de no más de una página dónde todos los integrantes del proyecto evalúan la experiencia de esta primera face (Retroalimentación al profesor)

### La entrega final deberá contener

1. Implementación de manejo de excepciones nativas y excepciones definidas por el usuario a partir de reglas de negocio, con uso de herencia y pruebas unitarias a los escenarios dónde se esperen dichas excepciones.
2. Manejo de archivos para guardar el estado de la aplicación.
3. Implementación de una interfaz gráfica de usuario para la aplicación. Esta puede ser de escritorio, web o móvil
4. Un informe de trabajo dónde quede en evidencia el rol de todos y cada uno de los miembros del equipo.
5. Completar el formulario individual de autoevaluación.
6. Un informe de no más de media página por estudiante con la evaluación que hace cada integrante del trabajo de sus compañeros (Coevaluación)
7. Un informe de no más de una página dónde todos los integrantes del proyecto evalúan la experiencia de esta primera face (Retroalimentación al profesor)

## Mecánica de trabajo grupal para todo el proyecto.

1. EL REPOSITORIO DEL PROYECTO: Cada grupo definirá un líder de proyecto. Este líder de proyecto creará un repositorio específico dentro de su github para que todos los miembros del proyecto realicen sus aportaciones a través de Pull requests.
2. LOS PULL REQUEST: Todos los Pull Request deberán ser revisados por al menos dos miembros del equipo antes de ser integrados al código base del proyecto, incluyendo los Pull Request del líder del proyecto. Toda verificación de Pull Request debe dejar un comentario así sea: Estoy de acuerdo con este Pull Request.
3. LA CARGA DE TRABAJO: El proyecto debe dividirse en segmentos de trabajo igualitarios, de manera que cada miembro del equipo tenga una carga de trabajo similar a la de los demás. Si el proyecto se supone tendrá 6 clases y el equipo es de 6 personas entonces cada persona deberá programar 1 clase e integrar su trabajo al proyecto.
4. ESTRUCTURA DE LAS RAMAS:
   1. Rama main o master. En esta rama estará la versión estable del proyecto. Solamente código estable, sin errores de compilación o ejecución deben vivir en esta rama
   2. Rama dev. En esta rama estarán integrados los cambios de todos los miembros del equipo. Sin importar que funcionen o no, siempre debe estar actualizada con la última versión del código de cada miembro del equipo. Una vez que la rama dev esté estable esta deberá integrarse a la rama main o master a través de un merge.
   3. Rama de trabajo. Cada miembro del equipo contará con una rama de trabajo. Esta rama deberá llamarse de la siguiente manera: apellido1\_apellido2\_nombre, por ejemplo la rama del profesor sería chacon\_vega\_jherom
   4. Todas las ramas deben estar en el repositorio del proyecto (luego del último commit siempre hacer push)
   5. Todos los miembros del equipo deben tener un fork del proyecto actualizado con la última versión a entregarle al profesor el día de la entrega en caso de que algo pase con el repositorio principal del proyecto definido en el punto 1
5. MECÁNICA DE LOS PULL REQUEST:
   1. Todo integrante del equipo sube sus cambios conforme los vaya desarrollando a su rama de trabajo que lleva su nombre
   2. Luego de subirla deberá hacer un pull request a la rama dev.
   3. Al menos dos miembros del equipo deben revisar el código y aprobarlo o declinarlo si ven que algo va mal
   4. Una vez están integrados los cambios a dev, la persona que solicitó el pull request deberá hacer pull a la rama dev en su computadora local y correrla en local para verificar que todo funciona bien y todos los unit test corren
   5. Si hay algún problema debe arreglarlo en su rama de trabajo y volver a reiniciar el proceso
   6. Una vez que la rama dev está estable y todo funciona incluyendo unit tests, deberán hacer un merge a main o master que será la rama que deberán presentar al profesor, quien hará un fork del repositorio al momento exacto de ser entregado por los estudiantes, generando una fotografía instantánea del código presentado para ser evaluado con objetividad.

## Los diagramas (14%)

Al ser menos diagramas que integrantes de cada equipo, pueden asignarse un diagrama por persona y revisarlos entre todos. Toda la programación debe ser consecuente con los diagramas. Es decir, deben programar las clases usando el diagrama de clases como base. La asignación de trabajo como equipo debe hacerse pensando en este diagrama también. Por lo que debe ser lo primero que hagan.

|  |  |  |  |
| --- | --- | --- | --- |
| DIAGRAMA DE CLASES 5% | | | |
| CRITERIO | No logrado | En proceso | Logrado |
| 1. Incluye todas las clases programadas en el proyecto |  |  |  |
| 1. Incluye todos los miembros de las clases |  |  |  |
| 1. Los nombres de las clases y miembros están de acuerdo a lo visto en clase |  |  |  |
| 1. Se demuestra el nivel de accesibilidad (public, private, protected) |  |  |  |
| 1. Se evidencian las relaciones de herencia y dependencia entre clases |  |  |  |
| DIAGRAMA DE CASOS DE USO 5% | | | |
| CRITERIO | No logrado | En proceso | Logrado |
| 1. Tiene marco y titulo |  |  |  |
| 1. Evidencia los actores |  |  |  |
| 1. Evidencia las actividades |  |  |  |
| 1. Evidencia las relaciones entre actividades y actores |  |  |  |
| 1. Evidencia las relaciones entre actividades |  |  |  |
| DIAGRAMA DE ESTADOS 2% | | | |
| CRITERIO | No logrado | En proceso | Logrado |
| 1. Evidencia los distintos estados de un objeto a lo largo de un proceso |  |  |  |
| 1. Tiene inicio y fin |  |  |  |
| DIAGRAMA DE FRECUENCIA 2% | | | |
| CRITERIO | No logrado | En proceso | Logrado |
| 1. Define la estructura vista en clase |  |  |  |
| 1. Sigue las líneas de los actores en sus determinados momentos |  |  |  |

## El informe de trabajo grupal **5%**

Debe evidenciarse en este informe todo el trabajo realizado por los miembros del equipo en la mecánica de trabajo descrita en el punto anterior. Debe explicarse que se le asignó a cada uno de los miembros del equipo y esta información debe ser consecuente con todo el historial de trabajo presente en el repositorio del proyecto. Es decir, si John Doe fue asignado a desarrollar la clase Persona y la clase Cliente entonces en el repositorio debe haber una rama para doe\_john con ese trabajo a modo de commit con fecha de realización (estos datos son automáticos cuándo se hace un commit). Este informe es muy breve, solo es una descripción de quien hizo que y dicha descripción debe ser un reflejo de lo que hay en el repositorio.

Recuerden que el repositorio nunca miente.

TODOS LOS MIEMBROS DEL EQUIPO DEBEN APORTAR PROGRAMACION. Nadie se puede quedar sin programar, ya que este es un curso de programación. Queda terminantemente prohibido que toda la programación le quede a una única persona o que toda la documentación le quede a única persona. Recuerden que hay tanta programación como documentación que hacer.

|  |  |  |  |
| --- | --- | --- | --- |
| INFORME GRUPAL 5% | | | |
| CRITERIO | No logrado | En proceso | Logrado |
| 1. Define una correcta distribución de cargas de trabajo |  |  |  |
| 1. Define el rol de cada miembro del equipo |  |  |  |
| 1. Define quien será el líder del proyecto |  |  |  |
| 1. Evidencia el trabajo presentado por el equipo |  |  |  |
| 1. Se ajusta a la extensión solicitada con el profesor siendo breve y conciso |  |  |  |

## Informe de coevaluación **5%**

En no más de un twit (250 caracteres) describir el desempeño de cada uno de los miembros del equipo según el punto de vista de cada quien.

Si John Doe trabajo con Jane Doe y con Martin Doe, entonces John Doe escribirá una breve reseña de lo que hizo Jane y lo que hizo Martin. Así Martin escribirá dos pequeñas reseñas de lo que hizo Jane y lo que hizo John. De la misma manera Jane escribirá una pequeña reseña de lo que hizo John y lo que hizo Martín. Junto a esta reseña deberán agregar una calificación de 1 a 5 dónde 1 es lo más bajo y 5 es lo más alto. Por ejemplo:

Esta es la reseña que John Doe hace de Jane Doe y Martin Doe:

1. Jane Doe es una gran programadora, muy metódica y sus pruebas unitarias son claras y exhaustivas. Calificación: 5/5
2. Martin Doe parece que aun no comprende bien la diferencia entre una clase abstracta y una interfaz. Casi no hace pruebas unitarias y de 10 métodos públicos probó solo 2. Calificación 2/5

LAS RESEÑAS DEBEN ENTREGARSE EN ORDEN ALFABÉTICO Y USANDO NUMERACIONES TAL Y COMO SE VUSIALIZA EN EL EJEMPLO

Esto deben hacerlo todos los miembros del equipo.

LAS RESEÑAS SON PERSONALES Y PRIVADAS, NADIE MAS QUE EL QUE LAS ESCRIBE Y EL PROFESOR QUE LAS LEE DEBEN CONOCERLAS.

|  |  |  |  |
| --- | --- | --- | --- |
| INFORME COEVALUACION 5% | | | |
| CRITERIO | No logrado | En proceso | Logrado |
| 1. Incluye a todos los miembros del equipo |  |  |  |
| 1. Asigna una calificación a cada miembro |  |  |  |
| 1. Se ajusta a la extensión indicada |  |  |  |
| 1. Evalúa temas relevantes al curso |  |  |  |
| 1. Se ajusta al formato solicitado por el profesor |  |  |  |

## Informe de reseña 6%

En conjunto todos los miembros del equipo deberán redactar en no más de una página su opinión sobre la experiencia. Pueden hacerlo de dos maneras, pueden ponerse de acuerdo y redactar juntos una sola reseña que plasme la opinión de todos, o pueden en una única página poner una breve opinión individual de lo que piensan cada uno de forma individual.

|  |  |  |  |
| --- | --- | --- | --- |
| INFORME RESEÑA 6% | | | |
| CRITERIO | No logrado | En proceso | Logrado |
| 1. Se refiere a la naturaleza grupal del proyecto |  |  |  |
| 1. Se refiere al uso del ciclo de trabajo con control de versiones |  |  |  |
| 1. Se refiere a la experiencia de trabajar los diagramas |  |  |  |
| 1. Se refiere a la experiencia de realizar los informes |  |  |  |
| 1. Se refiere a la experiencia de programar en equipo |  |  |  |
| 1. Se refiere a la carga que implica al estudiante trabajar un proyecto de esta forma vs lo que obtiene de la experiencia |  |  |  |

## Programación y repositorio 70%

|  |  |  |  |
| --- | --- | --- | --- |
| Programación 40% | | | |
| CRITERIO | No logrado | En proceso | Logrado |
| Conceptos de POO | | | |
| 1. Codifican todas las clases definidas en el diagrama de clases |  |  |  |
| 1. Codifican las interfaces definidas en el diagrama de clases |  |  |  |
| 1. Aplican la sobrecarga de constructores para instanciar objetos con atributos precargados |  |  |  |
| 1. Crean setters y getters como es debido |  |  |  |
| 1. Crea métodos privados en las clases |  |  |  |
| 1. Crean métodos públicos en las clases |  |  |  |
| 1. Crean métodos protegidos en las clases |  |  |  |
| 1. Crean paquetes para diferenciar cada parte del proyecto |  |  |  |
| 1. Usan clases abstractas |  |  |  |
| 1. Utilizan comportamientos o atributos estáticos |  |  |  |
| 1. Usan listas para crear colecciones de elementos |  |  |  |
| 1. Demuestran un correcto uso de la herencia para reutilizar código |  |  |  |
| 1. Utilizan la herencia para extender funcionalidades según se necesita |  |  |  |
| 1. Hacen un correcto uso de la palabra reservada this |  |  |  |
| 1. Hacen un correcto uso de la palabra reservada super |  |  |  |
| Pruebas unitarias | | | |
| 1. Crear un paquete para las pruebas unitarias |  |  |  |
| 1. Crean una clase de pruebas unitarias por cada clase desarrollada usando la misma estructura de paquetes |  |  |  |
| 1. Crean un método de prueba por cada método público de las clases desarrolladas |  |  |  |
| 1. Usan correctamente los métodos de comparación para las clases de pruebas |  |  |  |
| 1. Las pruebas unitarias pasan |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Repositorio 30% | | | |
| CRITERIO | No logrado | En proceso | Logrado |
| 1. Crean un repositorio principal para trabajar todos |  |  |  |
| 1. Crean una rama dev para integración |  |  |  |
| 1. Crean una rama por cada miembro del equipo |  |  |  |
| 1. Crean pull requests para integrar sus cambios |  |  |  |
| 1. Evidencian la revisión del código de sus pares |  |  |  |
| 1. Todos los pull request se verifican antes de aprobarse |  |  |  |
| 1. Todos los commits detallan que se está agregando |  |  |  |
| 1. Se evidencia el trabajo de todos los miembros del equipo |  |  |  |
| 1. Todos los miembros del equipo hacen fork antes de entregar el avance a un repositorio personal |  |  |  |
| 1. El código integrado en master o main es estable, compila y corre |  |  |  |
| 1. Se evidencia una carga equitativa del trabajo en equipo |  |  |  |
| 1. Se apegan a la estructura de nombres de ramas definidas en la mecánica de trabajo grupal. |  |  |  |
| 1. El branch main o master contiene todo el trabajo de todos los miembros |  |  |  |
| 1. Se evidencia todo el historial de trabajo en la rama main o master |  |  |  |
| 1. El repositorio del proyecto es accesible y puede hacerse fork |  |  |  |

# TEMAS

|  |  |
| --- | --- |
| Grupo | Tema |
| 1 | Punto de venta pulpería |
| 2 | Agenda de citas para servicios médicos |
| 3 | Inventario de productos |
| 4 | Servicios de taller |
| 5 | Organizador de tareas (To-do List) |
| 6 | Horarios de trabajo empleados |
| 7 | Registro de estudiantes |

|  |
| --- |
| 1. Punto de venta pulpería |
| 1. Agenda de citas para servicios médicos |
| 1. Inventario de productos |
| 1. Servicios de taller |
| 1. Organizador de tareas (To-do List) |
| 1. Horarios de trabajo empleados |
| 1. Registro de estudiantes |