UNIVERSIDADE DA CORUÑA

Diseño Software

Práctica 1 (2024-2025)

INSTRUCCIONES COMUNES A TODAS LAS PRÁCTICAS:

- **Grupos de prácticas**
  - Los ejercicios se realizarán preferentemente por parejas (pueden hacerse en solitario pero no lo recomendamos) y ambos miembros del grupo serán responsables y deben conocer todo lo que se entregue en su nombre. Recomendamos realizar la práctica con técnicas de "programación en pareja".
  - El nombre del equipo de prácticas será el nombre del grupo de prácticas al que pertenecen los miembros (con un prefijo "`DS-`") seguido por sus correspondientes *logins* de la UDC separados por guiones bajos, por ejemplo: `DS-12_jose.perez_francisco.garcia`[1].
  - En caso de pertenecer a grupos de prácticas distintos anteponer los dos grupos al inicio, siendo el primer grupo el que corresponde al primer login, como en el siguiente ejemplo: `DS-12-32_jose.perez_francisco.garcia`.
- **Entrega**
  - Los ejercicios serán desarrollados mediante la herramienta IntelliJ IDEA (versión *Community*) que se ejecuta sobre Java.
  - Los ejercicios se entregarán usando el sistema de control de versiones Git utilizando el servicio *GitHub Classroom*.
  - Tendremos una clase de prácticas dedicada a explicar Git, su uso en IntelliJ, GitHub Classroom y a cómo entregar las prácticas usando este sistema. Hasta entonces podéis ir desarrollando las prácticas en local.
  - Para la evaluación de la práctica sólo tendremos en cuenta aquellas contribuciones hechas hasta la fecha de entrega en el correspondiente repositorio de GitHub Classroom, los envíos posteriores no serán tenidos en cuenta.
- **Evaluación**
  - **Importante: Si se detecta algún ejercicio copiado en una práctica, ésta será anulada en su totalidad (calificación cero), tanto el original como la copia**.

[1] Si tenéis un *login* muy largo podéis tratar de acortarlo poniendo solo un apellido, siempre y cuando no exista confusión con algun compañero vuestro.

INSTRUCCIONES PRÁCTICA 1:

Fecha límite de entrega: 11 de octubre de 2024 (hasta las 23:59).

- **Realización de la práctica**
  - Los ejercicios se entregarán usando *GitHub Classroom*. En concreto en el *assignment* `2425-P1` del *classroom* `GEI-DS-614G010152425-OP1`.
  - Se deberá subir al repositorio un único proyecto IntelliJ IDEA para la práctica con el nombre del grupo de prácticas más el sufijo "`-P1`" (por ejemplo `DS-12_jose.perez_francisco.garcia-P1`).
  - Se creará un paquete por cada ejercicio de la práctica usando los siguientes nombres: `e1`, `e2`, etc.
  - Es importante que sigáis detalladamente las instrucciones del ejercicio, ya que persigue el objetivo de probar un aspecto determinado de Java y la orientación a objetos.
- **Comprobación de la ejecución correcta de los ejercicios con JUnit**
  - En la asignatura usaremos el framework JUnit 5 para comprobar, a través de pruebas, que el funcionamiento de las prácticas es el correcto.
  - En esta primera práctica os adjuntaremos algunos de los tests JUnit que deben pasar los ejercicios para ser considerados válidos.
  - **IMPORTANTE: No debéis modificar los tests que os pasemos**, sí podréis añadir nuevos tests para esos ejercicios si lo consideráis necesario. **Si un ejercicio no tiene tests es responsabilidad vuestra desarrollar los tests para el mismo**. Valoraremos la calidad de las pruebas realizadas.
  - En el seminario de JUnit os daremos información detallada de como ejecutar los tests y calcular la cobertura de los mismos y en el seminario de Git os comentaremos su integración con GitHub Classroom.
- **Evaluación**
  - Esta práctica corresponde a un 20 % de la nota final de prácticas.
  - **Criterios generales**: que el código compile correctamente, que no de errores de ejecución, que se hayan seguido correctamente las especificaciones, que se hayan seguido las buenas prácticas de la orientación a objetos explicadas en teoría, etc.
  - **Pasar correctamente nuestros tests es un requisito importante en la evaluación de esta práctica**.
  - Aparte de criterios fundamentales habrá criterios de corrección específicos que detallaremos en cada ejercicio.
  - No seguir las normas aquí indicadas significará una penalización en la nota.

1. **Manipulación de Strings**

   Crea una clase `StringGames` que implemente los siguientes tres métodos para proporcionar distintos juegos con `String`:

   - **bestCharacters**: dados dos `String` de igual longitud, determinar cuál gana en más categorías de las siguientes:
     - Más caracteres en minúscula.
     - Más caracteres en mayúscula.
     - Más caracteres de dígitos.

     Se devolverá aquel `String` que gane en más categorías, o el primero en caso de empate.
   - **crossingWords**: dados dos `String`, determinar de cuántas formas se podrían poner en un crucigrama; es decir, compartiendo una de sus letras.
   - **wackyAlphabet**: dados dos `String`, devolver el primero reordenado según el orden que indica el segundo, que debe contener todas las letras del alfabeto (y ningún otro caracter más).

   Ayuda:

   - Al ser juegos que no dependen de un estado, todos deberían ser definidos como públicos y estáticos (ejemplo: `public static String miMetodo(...))`.
   - La clase `Character` de Java tiene métodos que pueden ser útiles como `Character.isDigit(...)` o `Character.isLowerCase(...)`.
   - En el código de los tests, se encuentran ejemplos de cada uno de los juegos como comentarios, ilustrando cómo deben funcionar.

   Criterios:

   - Manejo de estructuras típicas de control de Java.
   - Manejo de las clases `String`, `StringBuilder` y sus métodos.
   - Manipulación de Strings paso a paso (no utilizar el API de colecciones de Java ni cosas que no se hayan explicado en clase como lambda-expresiones, expresiones regulares o streams).

2. **Código en un teclado alfanumérico**

Estamos diseñando un sistema de seguridad que consiste en teclear una serie de numeros en un teclado alfanumérico. El teclado tiene que cumplir los siguientes requisitos:

- Tener forma rectangular.
- El número en la esquina superior izquierda es el `1`.
- Los números siguen una secuencia bien por filas y luego columnas, o por columnas y luego filas.
- La secuencia que siguen es: primero los números (`1, 2, 3, .., 0`) y luego las letras mayúsculas usando el alfabeto inglés sin `Ñ`: (`A, B, .., Z`).

A continuación se muestran varios teclados válidos:

1	2	3
4	5	6
7	8	9

1	4	7	0	C
2	5	8	A	D
3	6	9	B	E

Para saber qué número hay que introducir en el teclado, en vez de decírselo directamente a la persona interesada, le diremos una secuencia de movimientos que tiene que hacer sobre el teclado partiendo de la esquina superior izquierda para llegar al número de la clave.

Los movimientos pueden ser arriba (`U`), abajo (`D`), izquierda (`L`) y derecha (`R`). Por ejemplo, si nuestra clave tiene cuatro dígitos le pasaremos los siguientes movimientos para cada uno de los números: `RD, DRUU, LLD, D`.

Estos movimientos sobre el primer teclado dan la clave `5347` de la siguiente forma:

- Empezamos en `1`, nos movemos a la derecha (`R`) y abajo (`D`) y llegamos al `5`.
- Partiendo del `5`, nos movemos abajo (`D`), a la derecha (`R`), arriba (`U`) y arriba (`U`) y llegamos al `3`.
- Partiendo del `3`, nos movemos a la izquierda (`L`), a la izquierda (`L`) y abajo (`D`) y llegamos al `4`.
- Partiendo del `4`, nos movemos abajo (`D`) y llegamos al `7`.

Si algún movimiento nos lleva fuera del teclado ignoraremos ese movimiento. De esta forma los movimientos `ULL, RRDDD, LURDL` sobre el primer teclado dan la clave de tres dígitos `198` de la siguiente forma:

- Empezamos en `1`, movimiento `U` ignorado, movimiento `L` ignorado, movimiento `L` ignorado, nos quedamos en `1`.
- Partiendo de `1`, nos movemos a la derecha (`R`), a la derecha (`R`), abajo (`D`), abajo (`D`) e ignoramos el último `D` llegando a `9`.
- Partiendo de `9`, nos movemos a la izquierda (`L`), arriba (`U`), a la derecha (`R`), abajo (`D`) y a la izquierda (`L`) llegando a `8`.

Dada esta definición del problema se pide desarrollar un código que implemente las siguientes funcionalidades:

- Determinar si un determinado teclado es válido y que cumple las condiciones especificadas anteriormente.
- Determinar si una secuencia de movimientos es válida, es decir, que son varios `String` conteniendo solo los caracteres `U`, `D`, `L` y `R`.
- Dado un teclado y una secuencia de movimientos, devolver la clave correspondiente a aplicar dicha secuencia sobre dicho teclado. Si el teclado o la secuencia no son válidos se debería lanzar una excepción de error.

Criterios:

- Manejo de estructuras típicas de control de Java.
- Manejo de *arrays* y matrices en Java.
- Lanzamiento de excepciones.

3. **Alquiler y compraventa de inmuebles**

   En este ejercicio deberéis crear clases, registros y enumerados de Java para representar anuncios de alquiler y compraventa de propiedades inmobiliarias.

   Nuevamente, tenéis ejemplos del funcionamiento en los tests que os incluimos.

   Una propiedad inmobiliaria se almacena en `Property` y un anuncio en `Ad`. Son, respectivamente, un registro de Java y una clase de Java. Aparte, habrá que crear enumerados de Java para ciertos tipos de datos en los que sea especialmente adecuado.

   `Property` redefinirá `toString` mostrando el catastro, el tipo de inmueble, la dirección (un String en formato libre), el código postal por separado, etc. El formato debe ser como se muestra a continuación:

```
APARTMENT
01234567890123456789
Aurelio Aguirre Galarraga 100, 1-A, A Coruna
15190
80 meters, 2 rooms, 1 bathrooms
```

   Asimismo, en `Property` se redefinirá el `equals` y el `hashCode`, con el criterio de que son iguales si y solo si el número de catastro es el mismo. Es posible que existan inmuebles "repetidos" que realmente son iguales, ya que el mismo inmueble puede estar en diferentes agencias con datos ligeramente distintos (por ejemplo, la dirección).

   Respecto a `Ad`, se almacenarán campos como la agencia, el tipo de anuncio, la `Property` correspondiente y el precio. Nuevamente, es normal que se repita el mismo inmueble en diferentes anuncios.

   Los métodos de `Ad` incluyen funcionalidades como comprobar si dos anuncios corresponden a inmuebles iguales (por igualdad, no necesariamente identidad), comprobar si el precio se sale de lo "normal" (los criterios los decidís, pero tenéis ejemplos en los tests), calcular el precio por m^2, rebajar el precio, etc. Se lanzarán excepciones para errores graves (nuevamente, hay ejemplos en los tests).

   Debéis completar los tests que os damos para asegurar una buena cobertura de código. Podéis añadir más tests, pero no podéis modificar los que os damos.

   Criterios:

   - Diferencia entre *igualdad* e *identidad*.
   - Contratos del `equals` y el `hashCode`.
   - Instanciación de objetos.
   - Constructores de copia.
   - Encapsulamiento.
   - Uso de *registros* y *enumerados*.
   - Compartir objetos inmutables con seguridad.
   - Manejo de *excepciones*.
   - Método `toString`.

4. **Euros**

   Una moneda de Euro se caracteriza por tener un valor nominal (un euro, 50 céntimos, etc.), un color (oro, bronce u oro-plata), un país, un diseño (representado aquí por simplificación como un `String`, por ejemplo, `"Juan Carlos I"` o `"Felipe VI"`) y un año de acuñación.

   En la actualidad los siguientes países acuñan monedas de Euro, identificados por su nombre (en inglés) y su código ISO de dos letras: *Andorra* (AD), *Austria* (AT), *Belgium* (BE), *Cyprus* (CY), *Croatia* (HR), *Estonia* (EE), *Finland* (FI), *France* (FR), *Germany* (DE), *Greece* (GR), *Ireland* (IE), *Italy* (IT), *Latvia* (LV), *Lithuania* (LT), *Luxembourg* (LU), *Malta* (MT), *Monaco* (MC), *Netherlands* (NL), , *Portugal* (PT), *San Marino* (SM), *Slovakia* (SK), *Slovenia* (SI), *Spain* (ES) y *Vatican City* (VA).

   Define la mejor estrategia para crear el registro `EuroCoin` haciendo uso de clases enumeradas como creas conveniente.

   Crea también una clase `EuroCoinCollection` que permita almacenar y mantener una colección de monedas de Euro. La clase nos permitirá introducir y borrar monedas de la colección, contar cuántas hay, saber su valor nominal total y comprobar si una moneda ya está en la colección.

   Si introducimos monedas repetidas en la colección estas no se añadirán. Se considera que dos monedas son repetidas si tienen el mismo valor, son del mismo país y tienen el mismo diseño visual (identificado este último como un `String`). El año de acuñación no se tiene en cuenta. Para evitar trabajar con números en coma flotante usaremos como unidad los céntimos de euro, de esta forma el valor de una moneda de dos euros es de 200 céntimos de euro.

Criterios:

- Tipos enumerados. Definición de tipos simples o complejos según convenga utilizando todas la información suministrada.
- Colecciones de objetos. Uso y selección de la colección de datos más apropiada para el problema.
- Uso de registros de Java.