PRÁCTICA 2 : Tareas *en Ada*

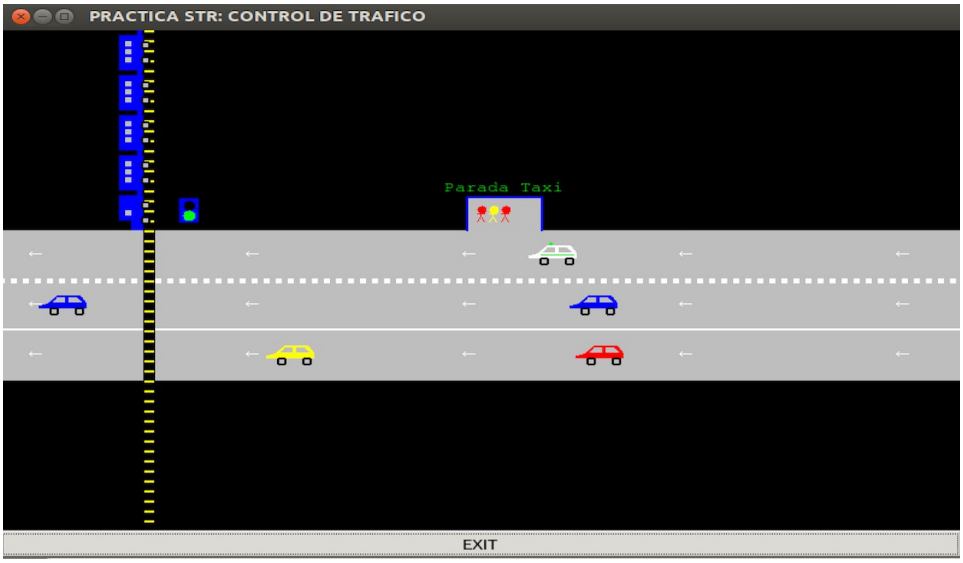
OBJETIVOS:

1. Conocer la representación de procesos concurrentes en el lenguaje Ada, implementando tareas para representar distintos elementos de un sistema simulado.
2. Detectar y comprender algunos de los problemas que surgen en la ejecución de un sistema concurrente.

PERIODO RECOMENDADO PARA SU REALIZACIÓN: 2 semanas

ENUNCIADO: Circulación de trenes y coches.

A partir del siguiente entorno gráfico y de un sistema de tareas concurrentes proporcionado para la aparición de taxis y de peatones, se debe implementar la circulación "simultánea" de coches y trenes. La siguiente figura muestra un ejemplo de la ejecución del sistema simulado completo en un instante determinado.



Tenemos tres carriles con el mismo sentido de circulación por los que circulan coches que siempre aparecen en el margen derecho de su carril. Por el carril superior circulan taxis y por los otros dos carriles el resto de coches. Ningún coche puede cambiar de carril.
El sentido de circulación de los trenes es de la parte superior a la parte inferior, por tanto todos los trenes deberán aparecer por el margen superior de la vía. El semáforo para regular el tráfico en esta práctica no tiene que funcionar.

Cada "objeto gráfico" individual que aparecerá en nuestro sistema (taxi, tren, peatón o coche) tiene que ser representado por una tarea distinta.

PASOS A SEGUIR:

En primer lugar, se detalla una breve descripción de los paquetes proporcionados:

- **Paquetes** que se deben usar como **ayuda** para implementar el proyecto y que en los enunciados de las prácticas, se irá indicando en cada caso qué usaremos de cada uno:
  - ***pkg_tipos*** : declaración de constantes y tipos de datos (**sólo el fichero .ads**)
  - ***pkg_graficos*** : procedimientos y funciones para la representación gráfica
- ***pkg_taxi***, ***pkg_peaton*** **: paquetes** que contiene la creación y el comportamiento de tareas que servirán de ejemplo para implementar nuevos paquetes en esta práctica para crear otras tareas:
- ***pkg_debug*** **: paquete** para imprimir en un fichero de texto y por pantalla mensajes para facilitar la **depuración** del programa si se considera necesario:
  - sólo necesitarás usar el procedimiento ***Escribir***
  - ¿Cómo se llama el fichero de texto donde se escriben los mensajes de salida?
  - ¿Por qué no tienes que invocar al procedimiento ***Crear_Fichero***?
- ***pkg_buffer_generico***, ***double_buffer*** : otros **paquetes** que sólo se necesitarán para la **compilación** del proyecto:

IMPORTANTE! **NO debes modificar** los paquetes proporcionados por el profesor, ya que se proporcionarán versiones con algunas modificaciones de algunos de ellos en posteriores prácticas. Por tanto, **deberas crear tus propios ficheros** en los que se definan paquetes que incluyan el código fuente propio que implementes.

PASO 1: Visualizar el entorno estático simulado

Crea un proyecto denominado **practica2** utilizando la plantilla "*Basic → Simple Ada Project*" y copia en su carpeta /src todos los paquetes proporcionados.
El cuerpo del programa principal que debes implementar debe contener lo siguiente:

```
begin
   pkg_graficos.Simular_Sistema;
end main;
```

Aclaración: el procedimiento *pkg_graficos.Simular_Sistema* se encarga de la animación gráfica. Implementa un bucle infinito que redibuja y captura eventos de la interfaz gráfica. Se interrumpe al pulsar el botón **Exit** de la ventana principal, acabando automáticamente la ejecución del programa principal y del resto de tareas en ejecución (si las hubiere).

Antes de compilar el proyecto, hay que añadir como dependencia del mismo la librería gráfica gtkAda. Para ello, desde el menú contextual del proyecto utiliza la opción *Project → Properties → Dependencies* y pulsando el botón '+' selecciona el fichero gtkada.gpr que se encuentra en la siguiente ruta de la máquina virtual: /usr/gnat/lib/gnat/
El campo "Limited with" no hay que activarlo

Al finalizar el paso 1, el resultado de la ejecución de tu proyecto deberá ser simplemente la visualización de la ventana principal del entorno simulado. Tu programa finalizará su ejecución cuando pulses el botón Exit.

PASO 2: Circulación de taxis

Entender la implementación proporcionada en el paquete ***pkg_taxi*** que permite la circulación de taxis. Para ello, **estudia el código** y responde las siguientes cuestiones para asegurarte que lo has entendido:

1. ¿Para qué se utiliza el tipo tarea *T_Tarea_Taxi?* ¿Cuántas instancias de tareas se crean de este tipo?
2. ¿Por qué crees que T*area_GeneraTaxis* se crea como una tarea anónima?
3. ¿Cuándo se crea cada tarea?
4. Describe el comportamiento de la tarea *Tarea_GeneraTaxis.* ¿Para qué se utiliza la constante *pkg_tipos.PERIODO_TAXI*? ¿Para qué se utiliza el tipo de datos *pkg_tipos.Ptr_T_RecordTaxi*?
5. ¿Cómo se actualizan los campos (atributos) del tipo T_RecordTaxi?
6. Describe el comportamiento de una instancia de tarea del tipo *T_Tarea_Taxi*

Para comprender el comportamiento de estas tareas, aquí tienes la descripción de los procedimientos y funciones utilizadas para la circulación de taxis y que están contenidas en el paquete ***pkg_graficos***:

- Las siguientes funciones específicas para taxis:
  - *Function* ***Hay_Taxi***: Devuelve *true* si existe actualmente un taxi en el carril. Es utilizada para no hacer aparecer un nuevo taxi cuando el carril todavía está ocupado
  - *Function* ***Pos_Parada_Taxi***: Devuelve *true* si el taxi está en la parada de peatones.
- **Los siguientes procedimientos y funciones sobrecargados**, que son comunes a cualquier tipo de "objeto gráfico" que aparece en el sistema:
  - *Procedure* ***Aparece***: Permite añadir un objeto gráfico representado en nuestro sistema (en este caso un taxi) al entorno simulado.
  - *Procedure* ***Actualiza_Atributo***: Actualiza el valor del atributo de un objeto gráfico (en este caso el atributo velocidad de un objeto taxi), y realizando el *casting* de una constante al tipo T_RangoVelocidad
  - *Procedure* ***Actualiza_Movimiento***: Actualiza la posición de un objeto gráfico, en esta caso del taxi especificado como parámetro, dependiendo de su velocidad.
  - *Function* ***Pos_Final***: Devuelve *true* si el objeto gráfico ha alcanzado su posición final dentro del sistema (ha terminado su circulación y por tanto debe desaparecer). En este caso, si el taxi especificado ha llegado al final de la carretera.
  - *Procedure* ***Desaparece***: Elimina del entorno simulado el objeto gráfico indicado (en este caso, el taxi).

Para simular el movimiento de los vehículos que aparecen en el sistema, se debe invocar al procedimiento pgk_graficos.Actualiza_Movimiento cada 0.1 segundos (valor de la constante *pkg_tipos.RETARDO_MOVIMIENTO*). Dependiendo de la velocidad del vehículo, el sistema calcula automáticamente su nueva posición.

Observa que al final del *body* de cada tarea hay un manejador de excepciones que nos permite capturar cualquier excepción no manejada en posibles bloques de código implementados dentro del *body* de la tarea e imprimir el nombre de la excepción lanzada. De esta forma, podemos conocer si una tarea ha acabado de forma "no esperada" y el motivo de su terminación "anormal".

Después de responder las cuestiones anteriores, incluye el paquete *pkg_taxi* en tu programa principal, es decir, debes añadir la siguiente línea:
with pkg_taxi;
Construye de nuevo tu proyecto (*Build*) y ejecútalo. ¿Por qué aparecen ahora los taxis circulando?

Al finalizar el paso 2, el resultado de la ejecución de tu proyecto deberá ser la aparición de un taxi cada 20 segundos (valor de la constante *PERIODO_TAXI*) que circulará por su carril a una velocidad constante. Se detendrá en la parada durante 2 segundos (valor de la constante TIEMPO_PARADA_OBLIGATORIA) y continuará circulando hasta desaparecer cuando llegue al final del carril. Es posible que en la ejecución no se refleje exactamente una velocidad constante, esto puede ser debido a la carga del procesador durante la ejecución y al hecho de utilizar una máquina virtual.

PASO 3: Aparición de peatones

Entender la implementación proporcionada en el paquete ***pkg_peaton*** que permite la aparición de peatones en la parada de taxis. Para ello, **estudia el código** y responde las siguientes cuestiones para asegurarte que lo has entendido:

1. ¿Cuántos peatones aparecen simultáneamente cada vez?
2. ¿Puede aparecer un grupo de peatones antes de que desaparezca el anterior?
3. ¿Qué colores pueden tener los peatones?
4. ¿A qué atributos de un peatón se le asigna un valor invocando al procedimiento Actualiza_Atributo antes de crear la tarea correspondiente?
5. ¿Por qué no se utiliza el procedimiento Actualiza_Atributo para inicializar los atributos de un taxi antes de crear la tarea correspondiente?
6. Describe el comportamiento de una instancia de tarea del tipo *T_Tarea_Peaton*

Del paquete pkg_tipos, para la aparición de peatones se utiliza lo siguiente:

- tipo ***Ptr_T_RecordPeaton***
- constante ***PERIODO_PEATONES*** (tiempo en segundos que determina la frecuencia de aparición de grupos de peatones)
- subtipo ***T_NumPeatones*** (utilizado en la generación del número aleatorio de peatones que formarán cada grupo que hacemos aparecer)
- subtipo ***T_ColorAparicionPeaton*** (utilizado en la generación de colores aleatorios para los peatones)

- tipo ***T_IdPeaton*** (utilizado como secuencia de identificadores de peatones, la secuencia debe inicializarse a **cero** cada vez que aparezca un grupo de peatones)
- constante ***TIEMPO_ESPERA_PEATON*** (tiempo en segundos que un peatón espera en la parada)

Del paquete pkg_graficos, utiliza los siguientes procedimientos y funciones:

- *Procedure* ***Aparece***
- *Procedure* ***Desaparece***
- *Procedure* ***Actualiza_Atributo***
- *Function* ***Parada_Taxi_Vacia*** devuelve true si la parada está vacía, en cuyo caso podremos hacer aparecer un nuevo grupo de peatones.
- *Function* ***Pos_inicio*** devuelve la posición en la parada del peatón según su identificador

PASO 4: Circulación de trenes

Crear un paquete con las funcionalidades necesarias para la circulación de trenes, que vienen dadas por los siguientes requerimientos:

1. Debe aparecer un nuevo tren con una determinada periodicidad.
2. Por la vía del tren no pueden circular simultáneamente varios trenes.
3. A cada tren se le debe asignar de forma aleatoria un número de vagones y un color.
4. Cuando el tren alcance el final de la vía, debe desaparecer.

Del paquete pkg_tipos, debes utilizar lo siguiente:

- constante ***PERIODO_TREN*** (tiempo en segundos que determina la frecuencia de aparición de los trenes)
- subtipo ***T_NumVagones*** (utilizado en la generación del número aleatorio de vagones con los que el tren debe aparecer)
- subtipo ***T_ColorTren*** (utilizado en la generación de colores aleatorios para los trenes)

Del paquete pkg_graficos, utiliza los siguientes procedimientos y funciones:

- *Procedure* ***Aparece***
- *Procedure* ***Desaparece***
- *Procedure* ***Actualiza_Atributo***
- *Function* ***Pos_Final***
- *Procedure* ***Actualiza_Movimiento***
- *Function* ***Hay_Tren***: Devuelve *true* si existe actualmente un tren en la vía. Es utilizada para no hacer aparecer un nuevo tren cuando la vía está ocupada por otro tren.

Al finalizar el paso 4, se debe haber añadido la siguiente funcionalidad a tu proyecto: la aparición con una frecuencia periódica de un nuevo tren con un color y un número de vagones aleatorio, que circulará a una velocidad constante hasta desaparecer cuando llegue al final de la vía.

PASO 5: Circulación de coches

Crear un paquete con las funcionalidades necesarias para la circulación de coches, que vienen dadas por los siguientes requerimientos:

1. Deben ir apareciendo nuevos coches en los dos carriles. **Cada carril debe estar representado por una tarea**, e irán apareciendo coches con una frecuencia de tiempo aleatoria. La secuencia aleatoria deberá ser distinta para cada carril.
2. Podrán circular coches simultáneamente por los 2 carriles.
3. A cada coche se le debe asignar (actualizar su correspondiente atributo):
    - el identificador de carril por el que debe circular
    - el identificador del coche, que será un número secuencial que lo identificará unívocamente en su carril (la secuencia tienes que comenzarla en 1 en cada carril)
    - un color aleatorio
    - una velocidad aleatoria. Si un coche tiene más velocidad que el de delante, no te debes preocupar de evitar que choque con él, ya que el sistema automáticamente regulará su velocidad para evitar colisiones.
    - la posición inicial en la que aparecerá el coche (usando la función Pos_Inicio)
4. El comportamiento de un coche de momento será muy simple: debe aparecer en el sistema, moverse con velocidad constante por su carril y finalmente cuando alcanza el final del carril, debe desaparecer del sistema, dando así por finalizada la correspondiente tarea.

Del paquete pkg_tipos, debes utilizar lo siguiente:

- tipo ***Ptr_T_RecordCoche***
- subtipo ***T_Carril*** (rango del número de carriles disponibles para los coches)
- subtipo ***T_RetardoCoches*** (utilizado en la generación de retardos aleatorios en la aparición de coches)
- subtipo ***T_ColorCoche*** (utilizado en la generación de colores aleatorios de coches)
- subtipo ***T_RangoVelocidadAparicion*** (utilizado en la generación aleatoria de velocidades iniciales de los coches)
- tipo ***T_IdCoche*** (utilizado como secuencia de identificadores de coches, inicializadas a **1** en cada carril)

Del paquete pkg_graficos, utiliza los siguientes procedimientos y funciones:

- Procedure ***Aparece***
- Procedure **Actualiza_Movimiento**
- Función **Pos_Inicio**
- Función **Pos_Final**
- Procedure **Desaparece**
- **Procedure *Actualiza_Atributo:*** recuerda que se utiliza para actualizar los distintos atributos que caracterizan un nuevo coche, es decir, su identificador, su color, su velocidad, el identificador del carril por el que circula y su posición inicial en el carril. **IMPORTANTE!!!** : éste último atributo se debe actualizar después de hacerlo con el identificador del carril

Al finalizar el paso 5, se deberá haber añadido la siguiente funcionalidad a tu proyecto: la aparición de coches con una frecuencia aleatoria en ambos carriles y con valores aleatorios de sus atributos color y velocidad, así como la circulación de dichos coches por sus respectivos carriles.