

Escuela Politécnica Nacional

Facultad de Ingeniería en Sistemas

Ingeniería en Ciencias de la Computación

Programación II



Semana 1

Clase 1: Martes, 30 de abril de 2024

El primer encuentro entre estudiantes, presentación habitual y pautas de las clases mostradas a continuación:

Datos generales

- ***Días de clase:*** Lunes/Martes/Miércoles
- ***Horario de clase:*** De 11h00 a 13h00
- ***Puntualidad:*** Se permite llegar hasta 10 minutos tarde.
- ***Profesor:*** Patricio Paccha (pat-mic)

Indicaciones (Pruebas-Exámenes-Proyecto)

- Las pruebas durarán de 2 a 4 horas
- Los exámenes durarán 8 horas
- El proyecto deberá tener una parte física, es decir, un aparato externo
- El proyecto deberá presentarse una parte al final del primer bimestre, y lo restante al final del semestre
- El markdown de cada estudiante deberá ser presentado al final de cada bimestre

Recursos de trabajo grupal

- Marcadores de pizarra
- Hojas de papel bond
- Pc's o celulares
- Pósits
- Taza de café (Opcional)
- Actitud: No trompud@, no brav@, no chuki

Normativas

- No llegar tarde a clases
- No hablar en el celular en clase, mantener en modo silencio
- Evitar comer alimentos en clase
- No distraerse en la computadora en actividades ajenas a la clase
- No entregar tareas, deberes, laboratorios, ni proyectos atrasados
- Subir las tareas a la plataforma virtual (o por correo)
- Pruebas teoricas en plataforma virtual (o escritas)
- Se tomará en cuenta la participación de eventos académicos que organice la Facultad de Sistemas

Sistema de evaluaciones

Evaluación	Puntaje	Temporalidad
Prueba	25%	Mensual
Examen	25%	Bimestral
Workshop	10%	Mensual
Homework	10%	Mensual
Proyecto	30%	Bimestral

Actuacion	+0.1	Siempre
Retor	+1.0	Siempre

Taller MA01

El taller consistió en realizar una torre de malvaviscos y fideos, los malvaviscos como uniones y los fideos como columnas/vigas, todo esto con la menor cantidad de recursos posibles e intentando obtener la mayor altura posible.

Clase 2: Miércoles, 1 de mayo de 2024

Personalización Visual Studio Code

- Se presentaron y añadieron extensiones útiles para el trabajo como markdownPDF, blackbox AI, gitLens y vscode-pdf.
- Se puede personalizar la terminal de PowerShell, git bash y la predeterminada de windows siguiendo el tutorial encontrado en la página OhMyPosh.

Comandos Linux

1. pwd : lugar actual
2. touch readme.md : crear archivos
3. code readme.md : abrir archivos
4. ls : listar archivos y carpetas
5. cd : cambia directorio de trabajo
6. mkdir : crea nuevo directorio
7. rm : elimina un archivo
8. cp : copiar archivos-directorios (incluido contenido)
9. mv : mueve o renombra archivos-directorios
10. file : comprueba tipo archivo
11. ls-la : lista ocultar archivos y directorios
12. nano, vi y jed : edita un archivo con un editor de texto
13. cat : lista, combina y escribe contenido de un archivo
14. sed : busca, sustituye o elimina patrones en un archivo
15. sort : reordena el contenido de un archivo
16. diff : compara el contenido de dos archivos
17. locate : busca archivos en la base de datos de un sistema
18. find : muestra ubicacion de un archivo o carpeta
19. useradd/userdel : crea y elimina cuenta de usuario
20. df : muestra uso general de espacio en disco
21. du : comprueba consumo almacenamiento de archivo o directorio
22. scp : copia de forma segura archivos o directorios a otro sistema
23. man : muestra el manual de un comando
24. echo : imprime un mensaje como salida estandar

Comandos VSCode

1. Ctrl + Shif + P : abre paleta de comandos
2. Ctrl + P : apertura rapida
3. Ctrl + B : abrir y cerrar el menu
4. Ctrl + D : cursor seleccion multiple
5. Shif + Alt + ↑ / Shif + Alt + ↓: copiar linea
6. Shif + Alt + A (comentario varia lineas) | Ctrl + K + C (comentario una sola linea) : Bloque de codigo de comentario
7. Alt + ←/→ : retroceder/avanzar
8. Ctrl + T : mostrar todos los simbolos
9. Ctrl + space | Ctrl + Shif + Space : sugerencia de activacion-parametros de activacion
10. Ctrl + Shif + N : abre ventana nueva
11. Ctrl + Shif + W : cierra una ventana
12. Ctrl + C : copia fragmento de codigo
13. Ctrl + X : corta fragmento de codigo
14. Ctrl + V : pega formato codigo copiado/cortado
15. Ctrl + +/- : aumentar tamaño de letra/disminuir
16. Ctrl + O : abre explorador de archivos
17. Ctrl + N : Crea un nuevo archivo
18. Ctrl + S : Guarda
19. Ctrl + F | Ctrl + H : abre buscador | sustituir texto
20. Ctrl + G : Desplazarnos a la linea que nos interesa

Uso de Markdown

- **Tipo de letra:**

- **palabras en negrita** : Con ** palabra-frase **
- *palabras en cursiva* : Con * palabra-frase *
- ***palabras en negrita y cursiva***: Con *** palabra-frase ***
- ==texto resaltado== : Con == palabra-frase ==
- ~~texto tachado~~: Con ~~ palabra-frase ~~


- **Listas:**

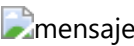
Lista no ordenada: Añadiendo - por cada item

- Elemento 1
- Elemento 2
- Elemento 3

Lista ordenada: Añadiendo 1.y 2. sucesivamente por cada item

- 1. Primer elemento
- 2. Segundo elemento
- 3. Tercer elemento

- **Enlaces:**
- [buscador google](#): Usando [nombre cualquiera] (link), sin espacio
- captura de codigo: Usando ![nombre cualquiera](nombre de la imagen)
- **Etiqueta:**



Con ![nombre cualquiera](link de la etiqueta)

Codigo Java

```
public class Hola {  
  
    // Clase principal de la aplicacion  
  
    public static void main(String[] args) {  
        System.out.println();  
    }  
}
```

Iniciando con ```NombreLenguajeDeProgramación y terminando con ```

Tabla

Columna 1	Columna 2
A	B
C	D

Usando |Nombre de columna 1|Nombre de columna n| /|---|---| Sin / |fila 1 columna 1| fila 1 columna n| |fila n columna 1| fila n columna n|

Bloques de codigo:

- comillas invertidas ---> ´codigo en linea´
 - citas ---> crear citas utilizando el signo ">"
 - Lineas horizontales:
-

- Guiones: -----
- Asteriscos: *****
- Guiones bajos: _____

Comandos de Git

Configurar identidad de Git/ Verificar

- git config --global user.name "nombre de usuario"
- git config --global user.email "correo electronico"
- git config user.name
- git config user.email
- touch NombreNuevoArchivo : Añade archivos al directorio actual

Control de versiones

- git init : Registra los archivos del directorio actual
- git status : Muestra los archivos sin guardar o no registrados por git
- git add . : Añade a lectura por git todos los archivos del directorio actual
- git commit -m "mensaje" : Guarda los cambios realizados con un comentario (mensaje)
- git push origin main: Envía a la nube (GitHub) los cambios realizados
- git log : Muestra los commits guardados

Inicio del control de versiones -clonando

- git clone linkGitHub
- git clone linkGitHub Nombre

Actualizar repositorio

- git pull origin main
- git fetch origin NombreRama: Descarga pero no actualiza/une los nuevos cambios del repositorio

Quitar archivos del control

- echo "nombreArchivo.ext >> .gitignore
- echo ".txt" >> .gitignore

Ver archivos ignorados

- cat .gitignore

Forzar agregar archivo excluido

- `git add -f NombreArchivo`

Branches/Ramas

- `git checkout -b NombreNuevaRama`: Crea una nueva rama en el repositorio
- `git checkout NombreRama`: Moverse a otra rama
- `git branch -d NombreRama`: Elimina una rama
- `git push origin NombreRama`: Manda los cambios a la nube, específicamente a la rama seleccionada
- `git checkout -M main`: Cambia el nombre de la rama principal a "main"

Merge/Union de Ramas

- `git merge NombreRama`: Une los cambios de la rama seleccionada a la rama actual
- `git diff Rama1 Rama2`: Mirar los cambios entre la rama 1 y la rama 2

Eliminar archivos

- `git rm NombreArchivo`
- `git rm -r`

Tagging

- `git tag NombreTag CodigoCommit`: Etiqueta un commit en específico (Para commits importantes)
- `git log`: Visualiza los ID's de los commits

Clave SSH

Utilizar primero: `ssh-keygen -t ed25519 -C "corre_electronico"` Luego, presionar enter hasta visualizar la frase: `The key's randomart image is:` Usar después, `eval "$(ssh-agent -s)"` para verificar que el agente ssh se este ejecutando. Luego, agregamos la llave privada SSH al ssh agent ejecutando: `ssh-add ~/.ssh/id_ed25519` Para visualizar la clave SSH ejecutamos: `cat id_ed25519.pub` Finalmente copiamos esta clave y nos dirigimos a GitHub, en Configuraciones buscar SSH and GPG keys y agregarla. Comprobar que la clave funciona ejecutando: `ssh -T git@github.com` Si observamos una advertencia como:

```
The authenticity of host "github.com(IP ADDRESS)" can't be established
ED25519 key fingerprint is SHA256: Clave.
Are you sure you want to continue connecting (yes/no)?
```

Escribir yes y observar que la clave funciona exitosamente.

Clase 3: Lunes, 6 de mayo de 2024***Programación orientada a objetos***

El primer lenguaje de programación para orientación a objetos nació en 1960, se llamó *Simula*.

Historia de Java

Sintaxis de Java

- **package** Son todos los archivos que pertenecen a un paquete, se coloca al inicio del archivo, es opcional
- **import** Importa los paquetes para el proyecto, es opcional, va después del package
- **public class Person**: Java usa clases para ejecutar el código
- **public static void main (String args[]){Código}**: Es la función principal de Java.
-

Programación estructurada Vs Programación orientada a objetos

- **UML**: Lenguaje de modelado unificado, fue presentado para diseñar con gráficos el sistema.

P.E	P.O.O
Struct	Clase
Variable	Variables/Propiedades-Atributo Las variables pertenecen a los métodos, si no son propiedades

Ejemplo de código en UML P.O.O.

```
public class Animal{
    public int id;
    private string nombre;
    protected char algo;
    friendly boolean algo;

    public static void main(){
        Animal a = new Animal; ///Declara una variable perteneciente a la clase Animal
    }
}
```

- Una diferencia de P.O.O. con P.E. es que se añade el ámbito, que va antes de declarar el tipo de dato, en este caso se tienen las más comunes public, private, protected y friendly. Van debajo de la clase usualmente, pueden ir debajo también. Si no se coloca el ámbito, en el caso de Java los declara automáticamente como private.

Ejemplo de código en UML P.E

```
#include <iostream>
#include <string>
struct Animal{
    string/char/etc Algo;
} type of animal;
```



```
void main(){
    animal *A = new Animal; ///Declara que se apunta al struct
    A->Algo = "algo"; ///Indica que se usará la memoria del puntero, no su direccion
    ...
}
```

Cuadro de UML general

Nombre de la Clase	Animal
Propiedades	+id: int +nombre: string
Metodo	
Eventos(Opcional)	+main(): void

- El simbolo + se usa en lugar de colocar el ambito public.
- El simbolo - se usa en lugar de colocar el ambito private.
- El simbolo # se usa en lugar de colocar el ambito protected.

POO: Metodo sin retorno

- Metodo que no devuelve ningun valor o cosa, es decir, un procedimiento.

POO: Metodo con retorno

- Metodo que retorna algun valor o cosa, es decir, es una funcion.

Semana 2

Clase 4: Lunes, 13 de mayo de 2024

P.O.O. : Conceptos, resolución de problemas

Sobre los objetos se pueden hacer dos cosas: Caracterizar (Propiedades) y Acciones (Métodos).

- **Propiedades:** Son los atributos que tiene un objeto, es decir, lo que describe a un objeto. También se puede ver como las características que tiene un objeto, se puede ver fácilmente estas graficando el objeto y describiéndolo.

Ejemplo:

El objeto es una mujer, las propiedades características pueden ser:

Género: 10, tipo de cabello: "Lacio", tipo de ojos: "verdes-alargados", estatura: 1,80, numero de brazos: 2, edad: 10, etc.

- **Métodos:** Son las acciones (Verbos en infinitivo) que puede realizar un objeto, es decir, lo que hace un objeto. Se realizan estos en base a las propiedades, es decir están directamente relacionadas a las **propiedades**. Estas acciones pueden o no tener **parámetros**. Los parámetros son similares a las propiedades, con la diferencia que los parámetros. Los métodos son similares a las funciones con la diferencia que estos tienen un ámbito (public: +, private: -, protected: ~, friendly: #, etc). Los métodos pueden retornar un valor o no,

Ejemplo:

```

Caminar(), Caminar(tiempo,lugar)
Dormir(tiempo, posición, lugar), respirar(), comer(tiempo, lugar, tipo)
Hablar( ), saltar(lugar, cantidad ), tocar(tiempo,lugar)
Bailar(canción, tipo, ritmo).

```

Ya definidas las propiedades y los métodos se puede definir el ámbito del objeto. Para el mismo objeto mujer.

Ejemplo:

```

-Edad: 14
+Genero: "mujer"
+Ojos: "verdes-alargados"
+NumeroDeBrazos: 3
public caminar(tiempo, lugar)
protected dormir(tiempo, posición, lugar)
private comer(tiempo, lugar, tipo)
public saltar(lugar, cantidad)
public tocar(tiempo, lugar)
friendly bailar(cancion, tipo, ritmo)

```

UML: Unified Modeling Language (Lenguaje de modelado unificado)

Dependiendo del objeto que estemos diseñando o modelando, se pueden utilizar solo propiedades, o solo acciones, o las dos. Modelando el objeto anteriormente visto, mujer, resulta: Para este caso, creamos una clase llamada **Hombre** que tenga un método **sentir(Mujer)** con un parámetro de objeto, para añadir algo al cuadro **Evento** del cuadro UML **MUJER**.

MUJER

```

- Edad: float
+ TipoDeCabello: String
+ TieneOjos: boolean

```

...

MUJER

~ Bailar (cancion: String, tiempo: int, ritmo: String): **String/void/boolean/...** (Retorno)

~ Tocar (tiempo: int, lugar: String): boolean

+ Saltar (lugar: String, cantidad: Int): void

...

~ Sentir (hombre: Hombre) : Puede tener o no retorno

Después de realizar el cuadro UML, se puede realizar el código en cualquier lenguaje P.O.O. Para esto, utilizaremos el cuadro de UML **MUJER**.

Ejemplo:

```
public class Mujer(){  
    private float Edad;  
    public boolean TieneOjos;  
    public String TipoDeCabello;  
    ...  
    protected String Bailar (String cancion, int tiempoMin){  
        ...  
        return "Bla bla bla";  
    }  
    ...  
}
```

==Deber: Conceptualizar un animal salvaje: 3 Propiedades/ 3 Métodos, Cuadro UML, Código. En cuaderno/hoja.==