UNNOBA
UNIVERSIDAD NACIONAL
NOROESTE | BUENOS AIRES

**IPOO – 2020**

**Nombre:**.....
**Apellido:**.....
**Legajo:**.....

Departamento de Informática y Tecnología **Parcial fecha única: 30/11/2020**

Se recomienda leer íntegramente el parcial antes de empezar a resolver el mismo. Los enunciados son claros y solo se contestarán preguntas relacionadas a los mismo, **nunca** a su solución. Utilizar solo las hojas provistas, en cada hoja, poner el nombre, apellido y número de alumno.

Puntos para aprobar el parcial: **60** puntos o más.

| Eje. | 1 | 2 | 3 | 4 | 5 | 6a | 6b | 6c | 6d | 7a | 7b | NOTA |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Respuesta** | | | | | | X | X | X | X | X | X | |
| **Resultado** | | | | | | | | | | | | |

## Ejercicio 1

¿Qué elementos crees que definen a un objeto?

(a) Sus cardinalidad y su tipo
(b) Sus atributos y sus métodos
(c) La forma en que establece comunicación e intercambia mensajes
(d) Su interfaz y los eventos asociados
(e) Ninguna de las anteriores.

## Ejercicio 2

¿Cuál de las siguientes declaraciones es válida en una interface?

(a) public double method();
(b) public final double method();
(c) public final void method(double d1);
(d) private void method(double d1);
(e) private String method();
(f) Ninguna de las anteriores

## Ejercicio 3

Dado el siguiente código Java:

```
public class Uno{

}
```

(a) Declara un objeto como público
(b) Deberia estar conteniudo en un archivo de texto llamado Uno.java
(c) Debería ser la única clase del archivo donde se encuentre
(d) ninguna de las anteriores.
(e) todas las anteriores.

## Ejercicio 4

¿Cuál es la salida del siguiente programa?

```
public class Prueba  {

    public static void main(String [] args)    {
        Prueba p = new Prueba();
        p.metodo();
    }

    void metodo()     {
        boolean b1 = false;
        boolean b2 = reemplaza(b1);
        System.out.println(b1 + " " + b2);
    }

    boolean reemplaza(boolean b1)    {
        b1 = true;
        return b1;
    }
}
```

(a) true true
(b) false true
(c) true false
(d) false false
(e) lanza una excepción en runtime.
(f) ninguna de las anteriores.

## Ejercicio 5

El **for-each** se utiliza para:

(a) Inicializar varias variables a la vez
(b) Borrar todos los elementos de una Colección
(c) Recorrer todos los elementos de una Colección
(d) Hacer la sumatoria de los elementos de la colección

## Ejercicio 6

Responda:

**a)** *En Java, ¿a qué nos estamos refiriendo si hablamos de **'Swing'**?*

**b)** *Explique que son los siguientes tipos de archivos en Java: **.java / .class / .jar***

**c)** *¿Qué es la **Sobrecarga de Métodos**?*

**d)** *¿A qué nos referimos cuando en Java hablamos de **circuito corto** y **circuito largo**?*

## Ejercicio 7

A continuación hay una parte de código que pretende emular el procesamiento de procesos y tareas de una **CPU** con varios **Cores**. Cada tarea (las cuales pueden ser diversas y contener diferente información) implementa la interface **Task**.

Se tiene una versión parcial de la implementación entregada. Es por ello que **se pide que usted, en base al código que está debajo:**

a) Realice el diagrama en UML del código a continuación.
b) Complete el código faltante. (Métodos, Clases, Interfaces, Declaraciones, etc.)

**Para ello copien el código en algún editor, completen y suban en formato ZIP con su nombre a la tarea correspondiente.**

**Nota**: Tenga en cuenta todas las clases, métodos y atributos que faltan.

```
public class CPU {

	private Cola cola = ____________________________________
	private List<Core> cores = ____________________________________

	//Procesa el siguiente proceso según la prioridad
	public void procesarNextProceso() {
		try {
			Proceso next = cola.getNext();
			getCore().procesar(next);
		} catch (____________________________________) {
			//Nada que hacer... el trabajo se ha completado
		}
	}

	private Core getCore() {
		return cores
				.stream()
				.filter(core -> !core.isOcupado())
				.findAny()
				.get();

	}

}

public class Core {

	private boolean ocupado;

	public void procesar(Proceso proceso) {
		setOcupado(true);
		proceso.procesarTodasLasTareas();
		setOcupado(false);
	}

	public boolean isOcupado() {
		return ocupado;
```

```
	}

	public void setOcupado(boolean ocupado) {
		this.ocupado = ocupado;
	}
}

public final class Cola {

	private List<Proceso> procesos = new ArrayList<Proceso>();

	private Comparator<Proceso> comparator = new Comparator<Proceso>() {
		@Override
		public int compare(Proceso o1, Proceso o2) {
			_____________________________________
		}
	};

	public Proceso getNext() throws NoHayProcesosException {

		if(procesos.isEmpty())
			_____________________________________

		return procesos
				.stream()
				.sorted(comparator)
				.findFirst()
				.get();
	}

}

public class Proceso {

	private Long id;
	private User owner;
	private Long prioridad;

	private List<Task> tasks;

	public Proceso(long id, User owner, long prioridad) {
		super();
		this.id = id;
		this.owner = owner;
		this.prioridad = prioridad;
		tasks = new ArrayList<Task>();
	}

	public Task getNext() _____________________________________{
		try {
			return tasks
					.stream()
					.findFirst()
					.get();
		}catch (NoSuchElementException e) {
			throw new NoHayTareasException();
		}
	}
```

```
	public void procesarTodasLasTareas() {
		tasks.forEach(task -> task.processed());
	}

	//Retorna TRUE si todas las tareas fueron procesadas. FALSE en caso contrario
	public boolean isProcessed(){
		_____________________________________________
	}

	public Long getPrioridad() {
		return prioridad;
	}

	public void setPrioridad(Long prioridad) {
		this.prioridad = prioridad;
	}

}

public interface Task {

	public double getDuracion();
	public String getName();
	public User getOwner();
	public boolean isProcessed();
	public void processed();

}

public class CopyFile implements Task {

    private String origen;
    private String destino;
    private long duracion;
    private boolean processed;

    public CopyFile(String origen, String destino, long duracion) {



        ___________________________________
    }

    @Override
    public double getDuracion() {
        return duracion;
    }

    @Override
    public String getName() {
        return origen + " -> " + destino ;
    }

    @Override
    public User getOwner() {
        return null;
    }
```

```
    @Override
    public boolean isProcessed() {
        return processed;
    }

    @Override
    public void processed() {
        this.processed = true;
    }
}

public class PlaySong implements Task {









}


public class NoHayProcesosException extends Exception {



        _____________________________________

}

public class NoHayTareasException extends Exception {

        private static final long serialVersionUID = 1L;

        @Override
        public String getMessage() {
                return _____________________________________
        }

}

public class Persona {

        private String nombre;
```

```
	private String DNI;
	private int edad;

	public Persona(String nombre, String dNI, int edad) {
		super();
		setNombre(nombre);
		setDNI(dNI);
		setEdad(edad);
	}

	public String getNombre() {
		return nombre;
	}

	public void setNombre(String nombre) {
		this.nombre = nombre;
	}

	public String getDNI() {
		return DNI;
	}

	public void setDNI(String dNI) {
		DNI = dNI;
	}

	public int getEdad() {
		return edad;
	}

	public void setEdad(int edad) {
		this.edad = edad;
	}

}

public class User extends Persona {

	private String nombreUsuario;
	private String password;

	public User(String nombre, String dNI, int edad, String nombreUsuario, String
password) {
		super(nombre, dNI, edad);
		this.setNombreUsuario(nombreUsuario);
		this.setPassword(password);
	}

	public String getNombreUsuario() {
		return nombreUsuario;
	}

	public void setNombreUsuario(String nombreUsuario) {
		this.nombreUsuario = nombreUsuario;
	}

	public String getPassword() {
		return password;
```

```
	}

	public void setPassword(String password) {
		this.password = password;
	}
}
```