Natalia Denise Fuentes Rubio
Semana 1 – Evidencias
Academia Java

Tarea 1: Realizar los simuladores *"Java Basics"* y *"Working with Java Data Types"* y entregar evidencias de cada simulador con calificación aprobatoria.

Java Basics

Name	Taken on - 19 jul, '24 12:03 PM	**Status**	Passed 100%
Correct Answers	16	**Total Questions**	16
Time Taken	00:10:13	**Total Time**	00:34:08
Start Time	19 jul 24 12:03	**Finish/Pause Time**	19 jul 24 12:13

Test Details | Performance Report

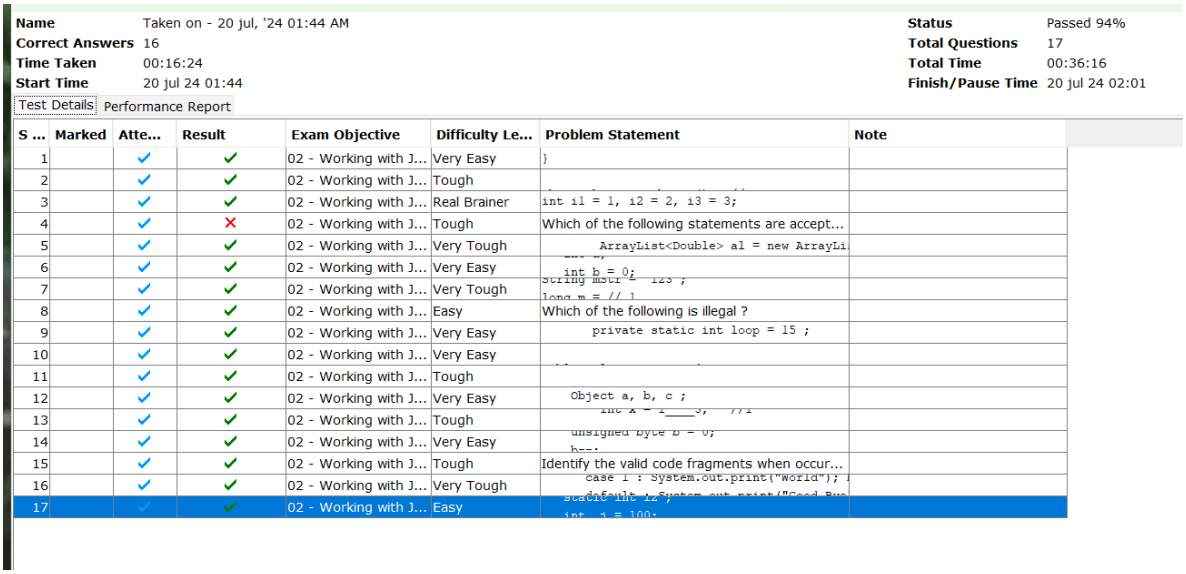
S ...	Marked	Atte...	Result	Exam Objective	Difficulty Le...	Problem Statement	Note
1		✓	✓	01 - Java Basics	Tough	// 1	
2		✓	✓	01 - Java Basics	Easy	}	
3		✓	✓	01 - Java Basics	Very Easy	if (args.length == 0){	
4		✓	✓	01 - Java Basics	Very Easy	package test;	
5		✓	✓	01 - Java Basics	Easy		
6		✓	✓	01 - Java Basics - O...	Tough	short firstValue = 5;	
7		✓	✓	01 - Java Basics	Very Easy	System.out.println(harry);	
8		✓	✓	01 - Java Basics - O...	Easy	}	
9		✓	✓	01 - Java Basics	Very Easy	System.out.println(args[1]);	
10		✓	✓	01 - Java Basics	Easy	int a; // (1)	
11		✓	✓	01 - Java Basics - O...	Easy	given the following requirements - 1. Implement three classes - Car, SUV, and	
12		✓	✓	01 - Java Basics - O...	Very Tough	Identify correct option(s)	
13		✓	✓	01 - Java Basics	Very Easy	Which method declarations will enable a class...	
14		✓	✓	01 - Java Basics	Very Easy	You have written some Java code in MyFirstClass.java file. Which of the following	
15		✓	✓	01 - Java Basics - O...	Very Easy	What is meant by "encapsulation" ?	
16		✓	✓	01 - Java Basics - O...	Easy	... other irrelevant code	

Name	Taken on - 19 jul, '24 11:31 AM	**Status**	Passed 82%
Correct Answers	14	**Total Questions**	17
Time Taken	00:19:44	**Total Time**	00:36:16
Start Time	19 jul 24 11:31	**Finish/Pause Time**	19 jul 24 11:51

Test Details | Performance Report

S ...	Marked	Atte...	Result	Exam Objective	Difficulty Le...	Problem Statement	Note
1		✓	✓	01 - Java Basics	Easy	Following options show the complete code lis...	
2		✓	✗	01 - Java Basics	Tough	contents of a file.	
3		✓	✓	01 - Java Basics	Easy	int a; // (1)	
4		✓	✓	01 - Java Basics	Very Easy	}	
5		✓	✓	01 - Java Basics - O...	Easy	Which of the following are features of Java?S...	
6		✓	✓	01 - Java Basics - O...	Easy	updateArea();	
7		✓	✓	01 - Java Basics - O...	Easy	Encapsulation ensures that ...	
8		✓	✓	01 - Java Basics	Very Easy		
9		✓	✓	01 - Java Basics	Very Easy	You have written some Java code in MyFirstClass.java file. Which of the following	
10		✓	✗	01 - Java Basics - O...	Tough	public void grow(int dy);	
11		✓	✓	01 - Java Basics	Tough	double x=10, double y; // 3	
12		✓	✗	01 - Java Basics	Tough	// 1	
13		✓	✓	01 - Java Basics - O...	Tough	short firstValue = 5;	
14		✓	✓	01 - Java Basics	Very Easy	if (args.length == 0){	
15		✓	✓	01 - Java Basics - O...	Easy	}	
16		✓	✓	01 - Java Basics	Very Easy		
17		✓	✓	01 - Java Basics - O...	Easy	When a class, whose members should be acc...	

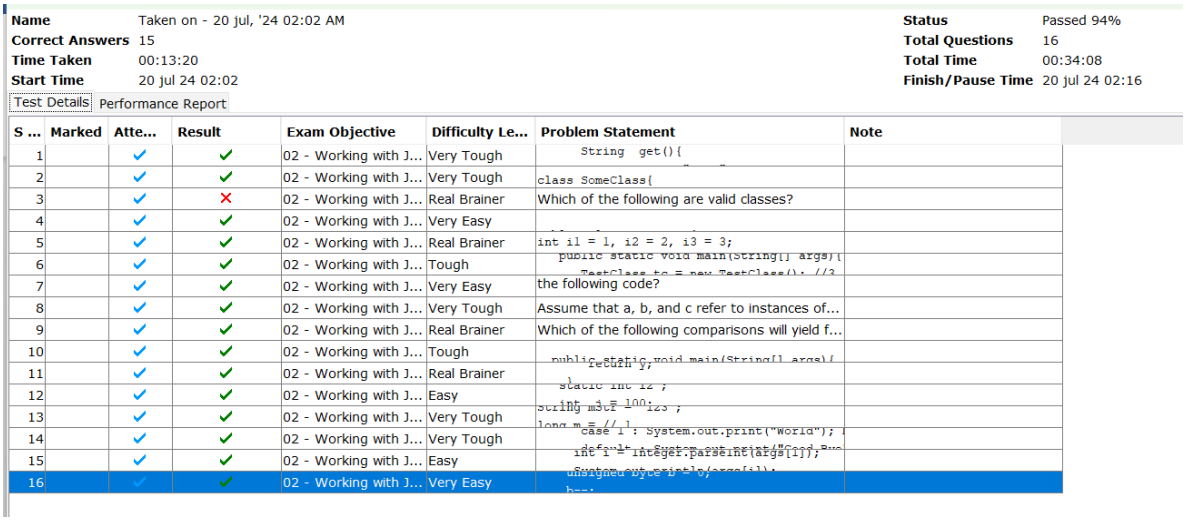
Working with Java Data Types



Name	Taken on - 20 jul, '24 01:44 AM	**Status**	Passed 94%
Correct Answers	16	**Total Questions**	17
Time Taken	00:16:24	**Total Time**	00:36:16
Start Time	20 jul 24 01:44	**Finish/Pause Time**	20 jul 24 02:01

Test Details | Performance Report

S ...	Marked	Atte...	Result	Exam Objective	Difficulty Le...	Problem Statement	Note
1		✓	✓	02 - Working with J...	Very Easy	}	
2		✓	✓	02 - Working with J...	Tough		
3		✓	✓	02 - Working with J...	Real Brainer	int i1 = 1, i2 = 2, i3 = 3;	
4		✓	✗	02 - Working with J...	Tough	Which of the following statements are accept...	
5		✓	✓	02 - Working with J...	Very Tough	ArrayList<Double> al = new ArrayLi	
6		✓	✓	02 - Working with J...	Very Easy	int b = 0;	
7		✓	✓	02 - Working with J...	Very Tough		
8		✓	✓	02 - Working with J...	Easy	Which of the following is illegal ?	
9		✓	✓	02 - Working with J...	Very Easy	private static int loop = 15 ;	
10		✓	✓	02 - Working with J...	Very Easy		
11		✓	✓	02 - Working with J...	Tough		
12		✓	✓	02 - Working with J...	Very Easy	Object a, b, c ;	
13		✓	✓	02 - Working with J...	Tough		
14		✓	✓	02 - Working with J...	Very Easy		
15		✓	✓	02 - Working with J...	Tough	Identify the valid code fragments when occur...	
16		✓	✓	02 - Working with J...	Very Tough		
17		✓	✓	02 - Working with J...	Easy		



Name	Taken on - 20 jul, '24 02:02 AM	**Status**	Passed 94%
Correct Answers	15	**Total Questions**	16
Time Taken	00:13:20	**Total Time**	00:34:08
Start Time	20 jul 24 02:02	**Finish/Pause Time**	20 jul 24 02:16

Test Details | Performance Report

S ...	Marked	Atte...	Result	Exam Objective	Difficulty Le...	Problem Statement	Note
1		✓	✓	02 - Working with J...	Very Tough	String get(){	
2		✓	✓	02 - Working with J...	Very Tough	class SomeClass{	
3		✓	✗	02 - Working with J...	Real Brainer	Which of the following are valid classes?	
4		✓	✓	02 - Working with J...	Very Easy		
5		✓	✓	02 - Working with J...	Real Brainer	int i1 = 1, i2 = 2, i3 = 3;	
6		✓	✓	02 - Working with J...	Tough	public static void main(String[] args){	
7		✓	✓	02 - Working with J...	Very Easy	the following code?	
8		✓	✓	02 - Working with J...	Very Tough	Assume that a, b, and c refer to instances of...	
9		✓	✓	02 - Working with J...	Real Brainer	Which of the following comparisons will yield f...	
10		✓	✓	02 - Working with J...	Tough		
11		✓	✓	02 - Working with J...	Real Brainer	return y;	
12		✓	✓	02 - Working with J...	Easy	static int i2 ;	
13		✓	✓	02 - Working with J...	Very Tough		
14		✓	✓	02 - Working with J...	Very Tough		
15		✓	✓	02 - Working with J...	Easy		
16		✓	✓	02 - Working with J...	Very Easy		

Tarea 2: Realizar un programa siguiendo el patrón de diseño *Singleton* documentado y dar una breve explicación.

Clase Gato

```
package com.singletonGato;

public class Gato {

	// Instancia de la clase Gato
	private static Gato;

	// Atributos de la clase
	private String nombre;
	private int peso;

	// Constructor privado para evitar instanciación externa
	private Gato(String nombre, int peso){
		this.nombre = nombre;
		this.peso = peso;
	}

	// Método público estático para la instancia
	public static Gato getInstance(String nombre, int peso) {
		if (gato==null)
			gato = new Gato(nombre, peso);
		return gato;
	}

	// Métodos de la clase
	public String getNombre() {
	  return nombre;
    }

    public int getPeso() {
        return peso;
    }

}
```

Clase Principal

```
package com.singletonGato;

public class Principal {

	public static void main(String[] args) {

		Gato gato1 = Gato.getInstance("Mayro", 7);
		Gato gato2 = Gato.getInstance("Salem", 6);

		//Verificación de que ambas variables apunten a la misma instancia
		//System.out.println(gato1 == gato2);
```

```
            // Llamada a métodos de la instancia singleton
            System.out.println("Tu gato se llama " + gato1.getNombre() + "," +
                                                " pesa " + gato1.getPeso() + "
kilos"+ "," +
                                                " tu gato esta pachoncito");

            System.out.println("Sé que tu gato no es " + gato2.getNombre() +
"," +
                               " y no pesa " + gato2.getPeso() + " kilos"+ "," +
                               " solo quiero que sepas que " + gato1.getNombre() +  "
está pachoncito");

      }

}
```

Breve explicación del código:

El método patrón de diseño Singleton, garantiza que tan solo exista un objeto de su tipo y proporciona un único punto de acceso a él para cualquier otro código.

Todas las implementaciones constan de:

- Hacer privado el constructor por defecto para evitar que otros objetos utilicen el operador *new.*
- Crear un método estático que actúe como constructor.

Para este ejemplo tenemos:

- El atributo estático *instance*, el cuál mantendrá la única instancia de la clase Gato.
- El constructor privado, con el fin de evitar que se puedan crear instancias adicionales desde fuera de la clase.
- El método *getInstance* de forma pública y estática, en este se verifica la existencia de la instancia si no, crea una nueva instancia y la devuelve. Si ya existe, simplemente devuelve la instancia existente.

Uso del Singleton: En el método *Principal*, se obtienen dos referencias al Singleton mediante *getInstance*. Se demuestra que ambas variables (gato1 y gato2) apuntan a la misma instancia, y se muestra cómo interactuar con los métodos de la instancia Gato.

Tarea 3: Realizar un programa implementando clases abstractas, interfaces, herencia y polimorfismo.

Clase Felino

```java
package com.PHCAI;

abstract class Felino {
    String nombre;

    Felino(String nombre) {
        this.nombre = nombre;
    }

    // Método abstracto
    abstract void hacerSonido();

    // Método no abstracto
    void dormir() {
        System.out.println(nombre + " está durmiendo.");
    }
}

      //Interface Cazador
      interface Cazador {
          void cazar();
      }

      // Interface Ronroneador
      interface Ronroneador {
          void ronronear();
      }
```

Felino es una clase abstracta, dentro de ella, se definen la estructura básica de nuestros felinos. Se definen los métodos, así como las interfaces para *Cazador* y *Ronroneador*.

Clase Gato

```java
package com.PHCAI;

class Gato extends Felino implements Ronroneador {

    Gato(String nombre) {
        super(nombre);
    }

    // Implementación del método abstracto
    @Override
    void hacerSonido() {
        System.out.println(nombre + " hace prrrrrrrrr");
    }
```

```
    // Implementación del método de la interface
    @Override
    public void ronronear() {
        System.out.println(nombre + " está ronroneando.");
    }
}
```

La clase gato, va a heredar de *Felino* e implementará la interfaz de *Ronroneador* implementando los métodos de *hacerSonido* y *ronronear*.

Clase León

```
package com.PHCAI;

class Leon extends Felino implements Cazador {

    Leon(String nombre) {
        super(nombre);
    }

    // Implementación del método abstracto
    @Override
    void hacerSonido() {
        System.out.println(nombre + " hace raawwwrrrr");
    }

    // Implementación del método de la interface
    @Override
    public void cazar() {
        System.out.println(nombre + " está cazando.");
    }
}
```

La clase León, va a heredar de *Felino* e implementará la interfaz *Cazador,* implementando los métodos *hacerSonido* y *cazar*.

Clase principal

```
package com.PHCAI;

public class Principal {
    public static void main(String[] args) {
        Felino gatito = new Gato("Salem");
        Felino gatote = new Leon("Simba");

        gatito.hacerSonido();
        gatito.dormir();

     // Uso del polimorfismo con la interface Ronroneador
        Ronroneador gatitoRonroneador = (Ronroneador) gatito;
        gatitoRonroneador.ronronear();
```

```
        gatote.hacerSonido();
        gatote.dormir();

        // Uso del polimorfismo con la interface Cazador
        Cazador gatoteCazador = (Cazador) gatote;
        gatoteCazador.cazar();
    }
}
```

En el método *Principal* se crean las instancias de *Gato* y *León*. Se implementa el *Polimorfismo* al definir a ambos como *Felinos* y también se muestra cómo el *Gato* puede ser *Ronroneador* al llamar al método *ronronear(),* y el *León* puede ser tratado como *Cazador* para llamar al método *cazar().*

Tarea 4: Dar una introducción breve a Git, sus comandos básicos, así como el trabajo en repositorios (branch, merge y conflicts)

Git

Es un sistema de control de versiones distribuido, el cual se puede definir como un software que ayuda a hacer un seguimiento de los cambios realizados a un código en el tiempo.

Sin el control de versiones, los desarrolladores suelen mantener una o más copias de diferentes versiones de algún código, lo que puede generar algún tipo de confusión que termine en la pérdida de información por error, ya sea cambiando el contenido de un archivo o su completa eliminación. Los sistemas de control de versiones solucionan este problema al administrar todas las versiones del código, presentando una versión a la vez a los participantes.

Dentro de las ventajas del control de versiones se encuentran la creación de flujos de trabajo, trabajo con versiones, la facilidad de codificar en equipo, mantener un historial y la automatización de tareas.

Git permite tener un clon local del proyecto es un repositorio de control de versiones completo. Dichos repositorios locales son plenamente funcionales y permiten trabajar sin conexión o de forma remota. Los desarrolladores trabajan localmente para posterior esto, sincronizar su copia del repositorio con la del servidor.

Esto lo distingue del control de versiones centralizado, donde se debe sincronizar el código con un servido antes de crear nuevas versiones.

Se puede decir que Git es un recurso sustentable, ya que la misma comunidad de usuarios ha creado recursos para enseñar su uso, manejo y ventajas por sobre otros métodos o servicios.

La mayoría de los entornos de desarrollo tienen compatibilidad con Git y su manejo desde la línea de comandos lo hace de fácil implementación en todos los sistemas operativos.

Comandos básicos de git

Comando	Función
git add nombre_de_archivo_aquí	Agregará un archivo al área de preparación
git add .	Agregará todos los archivos del proyecto al área de preparación
git status	Mostrará el estado del repositorio actual
git commit	Permitirá dejar un texto de confirmación y actualización en la terminal
git commit -m	Agregará un mensaje de confirmación sin abrir el editor
git log	Mostrará el historial de confirmaciones del repositorio
git diff	Mostrará los cambios realizados antes de confirmarlos
git checkout	Revertirá cambios no preparados
git reset HEAD nombre_del_archivo	Revertirá los cambios por etapas
git commit –amend	Modificará la confirmación más reciente
git revert	Revertirá la última confirmación
git branch nombre_de_la_rama	Creará una nueva rama
git checkout nombre_de_la_rama	Permitirá cambiar a otra rama
git branch	Mostrará todas las ramas creadas
git branch -d nombre_de_la_rama	Eliminará una rama
git merge nombre_de_otra_rama	Fusionará dos ramas
git merge –abort	Cancelará una fusión conflictiva
git add remote https://repositorio_aquí	Agregará un repositorio remoto
git remote show origin	Brindará más información sobre un repositorio remoto
git push	Enviará los cambios a un repositorio remoto
git pull	Extraerá los cambios de un repositorio remoto
git merge origin/main	Fusionará un repositorio remoto con el repositorio local
git push -u origin nombre_de_la_rama	Enviará una nueva rama a un repositorio remoto
git push --delete origin nombre_de_la_rama_aquí	Eliminará una rama remota
git init	Crea un nuevo repositorio

Trabajar repositorios en GitHub

Branches

Conforme se van realizando cambios de manera local al proyecto, se pueden actualizar en el repositorio remoto; con el fin de no alterar el proyecto final, se pueden realizar los cambios en rama personales o branches, ya que, de esta manera no se afectará el repositorio principal.

Una rama permite desarrollar características, corregir errores o experimentar de manera aislada de los cambios de terceros participantes del repositorio.

Se pueden crear a partir de una rama ya existente, se pueden crear desde la línea de comando en git, se guardan los archivos necesarios dentro de esta rama y se procede a realizarles las modificaciones necesarias. Una vez realizados los cambios se comparte la rama al repositorio. Una vez que se cerciore de que el código es correcto, se podrá unir a la rama principal del proyecto

Merge

Se le puede permitir a otros colaboradores con acceso al repositorio, fusionar sus solicitudes de incorporación de cambios en GitHub.
Para esto, se pueden configurar las opciones de fusión para una solicitud de extracción en GitHub, para que las necesidades del flujo de trabajo sean satisfactorias. Se puede implementar un método de fusión como el cabio de base o la combinación de confirmaciones.
El merge combinará múltiples secuencias de commits en un historial unificado. En el mayor de los casos, se fusionan dos ramas.

Conflictos
Sin embargo, al trabajar con ramas y realizando fusiones de las diferentes ramas creadas por todos los participantes, es normal que surjan conflictos que requieran resolución inmediata.
Usualmente estos conflictos se generan cuando dos, o más, desarrolladores han cambiado las mismas líneas en un archivo, o si un desarrollador elimino o modifico un archivo mientras otros lo estaban modificando de igual manera.
Para resolver el conflicto, es necesario editar manualmente el archivo en conflicto y decidir qué cambios se quieren mantener y eliminar aquellos que no sean necesarios.
Una vez resuelto, se puede reincorporar el archivo a la rama correspondiente y realizar la fusión.

Referencias

Blischak JD, Davenport ER, Wilson G (2016) A Quick Introduction to Version Control with Git and GitHub. PLoS Comput Biol 12(1): e1004668. https://doi.org/10.1371/journal.pcbi.1004668

https://docs.github.com/es/pull-requests/collaborating-with-pull-requests/addressing-merge-conflicts/about-merge-conflicts
https://docs.github.com/es/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-branches

¿Qué es Git? (s/f). Microsoft.com. Recuperado el 19 de julio de 2024, de https://learn.microsoft.com/es-es/devops/develop/git/what-is-git?source=recommendations

Rendón, M. (2023, marzo 15). *Hoja de referencia de Git: 50 comandos de Git que debes conocer*. freecodecamp.org. https://www.freecodecamp.org/espanol/news/hoja-de-referencia-de-git-50-comandos-de-git-que-debe-conocer-2/

Git - fundamentos de git. (s/f). Git-scm.com. Recuperado el 19 de julio de 2024, de https://git-scm.com/book/es/v2/Inicio---Sobre-el-Control-de-Versiones-Fundamentos-de-Git
https://gist.github.com/aaossa/7db152babead60ab097ba2c898d379a6
https://www.atlassian.com/git/tutorials/comparing-workflows