

# Simuladores

## Evidencia “Java Basics”

1.

Name Taken on - 17 jul, '24 11:53 AM  
Correct Answers 12  
Time Taken 00:44:18  
Start Time 17 jul 24 10:53

Status Passed 71%  
Total Questions 17  
Total Time 00:36:16  
Finish/Pause Time 17 jul 24 11:40

Test Details		Performance Report					
S ...	Marked	Atte...	Result	Exam Objective	Difficulty Le...	Problem Statement	Note
1		✓	✗	01 - Java Basics - O...	Easy	Which of the following are correct about "enc...	
2		✓	✓	01 - Java Basics - O...	Easy	Which of the following are benefits of polymo...	
3		✓	✗	01 - Java Basics - O...	Tough	short firstValue = 5;	
4		✓	✓	01 - Java Basics	Very Easy	Which of the following are valid declarations o...	
5		✓	✓	01 - Java Basics - O...	Easy	}	
6		✓	✓	01 - Java Basics - O...	Easy	given the following requirements -	
7		✓	✓	01 - Java Basics	Very Easy	1. Implement three classes - Car, SUV, and	
8		✓	✓	01 - Java Basics - O...	Easy	Which of the following are features of Java?S...	
9		✓	✗	01 - Java Basics	Tough	contents of a file.	
10		✓	✓	01 - Java Basics	Very Easy		
11		✓	✓	01 - Java Basics	Easy	value = Integer.parseInt(globa	
12		✓	✓	01 - Java Basics	Very Easy	if (args.length == 0 ){	
13		✓	✗	01 - Java Basics - O...	Easy	... other irrelevant code	
14		✓	✓	01 - Java Basics	Tough	public static long main(String[] args){	
15		✓	✓	01 - Java Basics	Very Easy	System.out.println("Hello");	

2.

6rs Test Overview Time Left - OVER LIMIT 00:02:32

Name

Taken on - 19 jul, '24 10:50 AM

Status

Passed 78%

Correct Answers 14

Total Questions 18

Time Taken 00:40:56

Total Time 00:38:24

Start Time 19 jul 24 10:50

Finish/Pause Time 19 jul 24 11:31

Test Details

Performance Report

S ...	Marked	Atte...	Result	Exam Objective	Difficulty Le...	Problem Statement	Note
1		✓	✓	01 - Java Basics	Very Easy		
2		✓	✗	01 - Java Basics - O...	Easy	Which of the following are benefits of polymo...	
3		✓	✗	01 - Java Basics	Tough	and	
4		✓	✓	01 - Java Basics	Easy	}	
5		✓	✓	01 - Java Basics	Very Easy	System.out.println(harry);	
6		✓	✓	01 - Java Basics	Very Easy	if (args.length == 0 ){	
7		✓	✓	01 - Java Basics	Very Easy	you have written some java code in	
8		✓	✗	01 - Java Basics	Very Easy	MyFirstClass.java file. Which of the following	
9		✓	✓	01 - Java Basics	Very Easy	System.out.println(args[1]);	
10		✓	✗	01 - Java Basics - O...	Very Tough	Identify correct option(s)	
11		✓	✓	01 - Java Basics - O...	Easy	Which of the following are correct about "enc...	
12		✓	✓	01 - Java Basics - O...	Easy	public void setradius(int r){	
13		✓	✓	01 - Java Basics - O...	Easy	given the following requirements -	
14		✓	✓	01 - Java Basics - O...	Easy	1. Implement three classes - Car, SUV, and	
15		✓	✓	01 - Java Basics	Tough	Which of the following are features of Java?S...	

<

View Questions

Pause Test

Evaluate Test

>

## **Evidencia “Working with Java Data Types”**

**1.**

## **Git & GitHub**

### **Introducción**

Hoy en día la colaboración es fundamental en el desarrollo de software, sitios web, y entre otros tipos de proyectos, por lo que se tiene la necesidad de trabajar sobre un mismo proyecto con agilidad y eficiencia. Nosotros podemos tener archivos en nuestro equipo y llevar un control sobre sus cambios, sin embargo, podríamos errar al entrar a la última versión, además se le suma la necesidad de introducir los cambios realizados por todo el equipo. Derivado de esta problemática surge lo que se presenta a continuación.

### **Sistema local de control de versiones.**

Tiene una base de datos que mantiene todos los cambios en los archivos bajo control de revisión. Uno de estos sistemas es RCS (Revision Control System) que gestiona múltiples revisiones de archivos. RCS automatiza el almacenamiento, recuperación, registro, identificación y combinación de revisiones.

### **Sistema de control de versiones centralizado.**

Los sistemas de control de versiones centralizados contienen solo un repositorio a nivel mundial y cada usuario debe comprometerse a reflejar sus cambios en el repositorio. Es posible que otros vean sus cambios mediante la actualización. Una ventaja es que todos saben hasta cierto punto lo que están haciendo todos los demás en el proyecto. Los administradores tienen un control detallado sobre quién puede hacer qué, y es mucho más fácil administrar un CVCS que tratar con bases de datos de cada cliente.

### **Sistemas de control de versiones distribuidos.**

Los sistemas de control de versiones distribuidos contienen múltiples repositorios. Cada usuario tiene su propio repositorio, así como una copia de trabajo. El solo hecho de confirmar sus cambios no dará acceso a otros colaboradores a estos. Esto se debe a que la confirmación reflejará esos cambios en el repositorio local y deberá enviarlos para que sean visibles en el repositorio central. De manera similar, cuando actualiza, no obtiene los cambios de otros a menos que primero haya ingresado esos cambios en su repositorio.

Los sistemas más populares de Sistemas de control de versiones distribuidos es Git y Mercurial.

### **Git**

Git, que presenta una arquitectura distribuida, es un ejemplo de DVCS (sistema de control de versiones distribuido, por sus siglas en inglés). En lugar de tener un único espacio para todo el historial de versiones del software, como sucede de manera habitual en los sistemas de control de versiones antaño populares, como CVS o Subversion (también conocido como SVN), en Git, la copia de trabajo del código de cada desarrollador es también un repositorio que puede albergar el historial completo de todos los cambios.

Funcionamiento Git

## Configuración de un repositorio comandos

### 1. Git init

- a. `$ git init` – crea un nuevo repositorio de Git. Puede utilizarse para convertir un proyecto existente y sin versión en un repositorio de Git, o para inicializar un nuevo repositorio vacío.
- b. `$ git clone <repo url>` - se emplea para crear una copia de un repositorio ya existente, git clone llama primero a git init par generar un nuevo repositorio, luego copia los datos del repositorio existente y extrae un nuevo conjunto de archivos de trabajo
- c. `$ git init --bare <directory>` - Inicializa un repositorio de git vacío, pero omite el directorio de trabajo. Los repositorios compartidos deberían crearse con la marca `--bare`.
- d. `$ git init <directory> --template=<template_directory>` - Las plantillas te permiten inicializar un nuevo repositorio con un subdirectorio de `.git` predefinido. Puedes configurar una plantilla para que tenga los directorios y archivos predeterminados que se copiarán en el subdirectorio de `.git` del nuevo repositorio.

### 2. Git clone

- a. `git clone` es una utilidad de línea de comandos de Git que se utiliza para fijar como objetivo un repositorio existente con el fin de clonarlo o copiarlo. Una vez que un desarrollador ha obtenido una copia de trabajo, todas las operaciones de control de versiones se gestionan por medio de su repositorio local. La clonación crea automáticamente una conexión remota llamada "origin" que apunta al repositorio original.
- b. `$ git clone <repo> <directory>` - Clona el repositorio ubicado en `<repo>` en la carpeta llamada `~<directory>!` en la máquina local.
- c. `$ git clone --branch <tag> <repo>` - Clona el repositorio ubicado en `<repo>` y clona solamente la referencia para `<tag>`
- d. `git clone -depth=1 <repo>` - Clona el repositorio ubicado en `< repo >` y clona solamente el historial de confirmaciones especificado por la opción `depth=1`. En este ejemplo, se realiza una clonación de `< repo >` y solo se incluye la confirmación más reciente en el nuevo repositorio clonado. La clonación superficial es muy útil cuando se trabaja con repositorios que tienen un largo historial de confirmaciones.

- e. `$ git clone --branch` - El argumento `--branch` permite especificar una rama concreta para clonarla en vez de la rama a la que apunta el HEAD remoto, normalmente la rama principal. Asimismo, puedes incluir una etiqueta en vez de una rama con el mismo efecto.

- f.

### 3. Git config

- a. El caso práctico más básico de git config es invocarlo con un nombre de configuración, que mostrará el valor definido con ese nombre. Los nombres de configuración son cadenas delimitadas por puntos que se componen de una "sección" y una "clave" en función de su jerarquía. Por ejemplo: `user.email`.

- b. Niveles y archivos de git config

- `--local` – aplica al repositorio de contexto en el que se invoca git config
- `--global` – aplica al usuario de un sistema operativo
- `--system` – afecta a todos los usuarios de un sistema operativo y a todos los repositorios.
- Ejemplo: `$ git config --global user.email "your_email@example.com"`

- c. a

### 4. Git commit

- a.

### 5. Git diff

- a.

### 6. Git stash

- a.

### 7. .gitignore