SIMULADORES

Using Operators and Decision Constructors

1.

Test Overview Time Left - 00:09:00

Name	Taken on - 26 jul, '24 12:28 AM	**Status**	Passed 88%
Correct Answers	14	**Total Questions**	16
Time Taken	00:25:08	**Total Time**	00:34:08
Start Time	26 jul 24 00:28	**Finish/Pause Time**	26 jul 24 00:54

Test Details Performance Report

S ...	Marked	Atte...	Result	Exam Objective	Difficulty Le...	Problem Statement	Note
1		✓	✓	03 - Using Operator...	Real Brainer	Object t = new Integer(107);	
2		✓	✗	03 - Using Operator...	Real Brainer		
3		✓	✓	03 - Using Operator...	Very Easy	3.	
4		✓	✗	03 - Using Operator...	Very Easy	boolean flag = true;	
5		✓	✓	03 - Using Operator...	Tough	case 2:	
6		✓	✓	03 - Using Operator...	Real Brainer	System.out.println(true + null); //2	
7		✓	✓	03 - Using Operator...	Easy		
8		✓	✓	03 - Using Operator...	Very Easy	Object obj1 = new Object(); Object obj2 = obj1;	
9		✓	✓	03 - Using Operator...	Easy	default : System.out.println(	
10		✓	✓	03 - Using Operator...	Easy	case true :	
11		✓	✓	03 - Using Operator...	Tough	System.out.println("Hello!");	
12		✓	✓	03 - Using Operator...	Easy	Which of the following statements are true?	
13		✓	✓	03 - Using Operator...	Very Easy	int j = 1; if((i++ == 0) & (i++ == 2)){	
14		✓	✓	03 - Using Operator...	Very Easy	while(checkit(k)){	
15		✓	✓	03 - Using Operator...	Easy	double d = 3.8;	
16		✓	✓	03 - Using Operator...	Very Easy	int a = 1;	

2.

Test Overview Time Left - 00:24:33

Name	Taken on - 26 jul, '24 12:55 AM	**Status**	Passed 80%
Correct Answers	16	**Total Questions**	20
Time Taken	00:18:07	**Total Time**	00:42:40
Start Time	26 jul 24 00:55	**Finish/Pause Time**	26 jul 24 01:13

Test Details Performance Report

S ...	Marked	Atte...	Result	Exam Objective	Difficulty Le...	Problem Statement	Note
1		✓	✓	03 - Using Operator...	Very Easy	while(checkit(k)){ System.out.print(k);	
2		✓	✗	03 - Using Operator...	Very Easy	case 1:	
3		✓	✓	03 - Using Operator...	Tough	static int x = 5;	
4		✓	✓	03 - Using Operator...	Very Easy	int a = 1;	
5		✓	✓	03 - Using Operator...	Easy	case true :	
6		✓	✓	03 - Using Operator...	Easy	int i;	
7		✓	✓	03 - Using Operator...	Real Brainer	else //4	
8		✓	✓	03 - Using Operator...	Real Brainer	System.out.println(true + null); //2	
9		✓	✗	03 - Using Operator...	Real Brainer	int[] a = { 1 };	
10		✓	✓	03 - Using Operator...	Tough	System.out.println("Hello!");	
11		✓	✓	03 - Using Operator...	Easy	throws Exception {	
12		✓	✓	03 - Using Operator...	Easy		
13		✓	✗	03 - Using Operator...	Real Brainer		
14		✓	✓	03 - Using Operator...	Very Easy	3.	
15		✓	✓	03 - Using Operator...	Very Tough	}	
16		✓	✗	03 - Using Operator...	Very Tough	switch(x){ case 'b': // 1	
17		✓	✓	03 - Using Operator...	Very Easy	Which of the following implementations of a max() method will correctly return the largest	
18		✓	✓	03 - Using Operator...	Tough	}	
19		✓	✓	03 - Using Operator...	Tough	case 2:	
20		✓	✓	03 - Using Operator...	Real Brainer	String str1 = "one";	

Creating and Using Arrays

1.

Test Overview Time Left - OVER LIMIT 00:20:16

Name	Taken on - 26 jul, '24 11:12 AM	**Status**	Passed 78%
Correct Answers	7	**Total Questions**	9
Time Taken	00:39:28	**Total Time**	00:19:12
Start Time	26 jul 24 11:12	**Finish/Pause Time**	26 jul 24 11:55

Test Details Performance Report

S ...	Marked	Atte...	Result	Exam Objective	Difficulty Le...	Problem Statement	Note
1		✓	✓	04 - Creating and U...	Very Easy	Which of the following statements about an a...	
2		✓	✗	04 - Creating and U...	Very Easy	`incr(i) ;`	
3		✓	✓	04 - Creating and U...	Easy		
4		✓	✓	04 - Creating and U...	Very Easy	`//1 : insert Line of code here`	
5		✓	✗	04 - Creating and U...	Easy	`FunWithArgs fwa = new`	
6		✓	✓	04 - Creating and U...	Very Easy	Which of these array declarations and instant...	
7		✓	✓	04 - Creating and U...	Easy	`initialize`	
8		✓	✓	04 - Creating and U...	Very Easy	Which of the following option(s) correctly dec...	
9		✓	✓	04 - Creating and U...	Very Easy	`boolean[] bA = new boolean[];`	

2.

Test Overview

Name	Taken on - 27 jul, '24 05:12 PM	**Status**	Passed 67%
Correct Answers	6	**Total Questions**	9
Time Taken	00:27:26	**Total Time**	00:19:12
Start Time	27 jul 24 17:12	**Finish/Pause Time**	27 jul 24 17:42

Test Details Performance Report

S ...	Marked	Atte...	Result	Exam Objective	Difficulty Le...	Problem Statement	Note
1		✓	✓	04 - Creating and U...	Very Easy	`boolean[] bA = new boolean[];`	
2		✓	✗	04 - Creating and U...	Easy	`int i = 4;`	
3		✓	✓	04 - Creating and U...	Very Easy	`//1 : insert Line of code here`	
4		✓	✗	04 - Creating and U...	Very Easy	`incr(i) ;`	
5		✓	✓	04 - Creating and U...	Tough	`){`	
6		✓	✗	04 - Creating and U...	Tough	Which of the following are valid code fragmen...	
7		✓	✓	04 - Creating and U...	Tough	`matrix[0][0] = "petrol";`	
8		✓	✓	04 - Creating and U...	Very Easy	Which of the following option(s) correctly dec...	
9		✓	✓	04 - Creating and U...	Real Brainer	`int[] a = { 1, 2, 3, 4 };`	

Using Loop Constructors

1.

Test Overview Time Left - OVER LIMIT 00:41:44

Name	Taken on - 26 jul, '24 02:11 PM	**Status**	Passed 80%
Correct Answers	8	**Total Questions**	10
Time Taken	01:03:04	**Total Time**	00:21:20
Start Time	26 jul 24 14:11	**Finish/Pause Time**	26 jul 24 15:1

Test Details Performance Report

S ...	Marked	Atte...	Result	Exam Objective	Difficulty Le...	Problem Statement	Note
1		✓	✗	05 - Using Loop Co...	Tough	`int i = 0;`	
2		✓	✓	05 - Using Loop Co...	Easy	`j++){`	
3		✓	✓	05 - Using Loop Co...	Easy		
4		✓	✓	05 - Using Loop Co...	Very Easy	Which of these statements are valid when oc...	
5		✓	✓	05 - Using Loop Co...	Easy		
6		✓	✗	05 - Using Loop Co...	Tough	`int i;`	
7		✓	✓	05 - Using Loop Co...	Very Easy	`if(count % 3 == 0) continue;`	
8		✓	✓	05 - Using Loop Co...	Easy		
9		✓	✓	05 - Using Loop Co...	Very Tough	`int x = 10;`	
10		✓	✓	05 - Using Loop Co...	Very Easy	`public static void main(String[]`	

2.

Test Overview Time Left - OVER LIMIT 00:09:07

Name	Taken on - 27 jul, '24 06:08 PM	**Status**	Passed 78%
Correct Answers	7	**Total Questions**	9
Time Taken	00:28:19	**Total Time**	00:19:12
Start Time	27 jul 24 18:08	**Finish/Pause Time**	27 jul 24 18:4

Test Details Performance Report

S ...	Marked	Atte...	Result	Exam Objective	Difficulty Le...	Problem Statement	Note
1		✓	✓	05 - Using Loop Co...	Tough	for(String day : days){	
2		✓	✓	05 - Using Loop Co...	Easy	You have been given an array of objects and ...	
3		✓	✗	05 - Using Loop Co...	Tough	do {	
4		✓	✗	05 - Using Loop Co...	Real Brainer	if (j > i) break labelB;	
5		✓	✓	05 - Using Loop Co...	Very Easy	int i;	
6		✓	✓	05 - Using Loop Co...	Tough	Which of the following are true about the enh...	
7		✓	✓	05 - Using Loop Co...	Very Easy	public static void main(String[] args)	
8		✓	✓	05 - Using Loop Co...	Very Easy	while(x<values.length){	
9		✓	✓	05 - Using Loop Co...	Very Easy	for (int i = 0, j = 10; sum > 20; ++	

SPRING BATCH

Introducción

Batch hace referencia a lotes por lo que el procesamiento por lotes consiste en aquellos programas que se lanzan generalmente de manera programada y que no requieren ningún tipo de intervención humana. Los cuales se caracterizan por ser procesos relativamente pesados, que tratan una gran cantidad de información, lo que hace que se ejecuten en horario con baja carga de trabajo para no influir en el entorno transaccional.

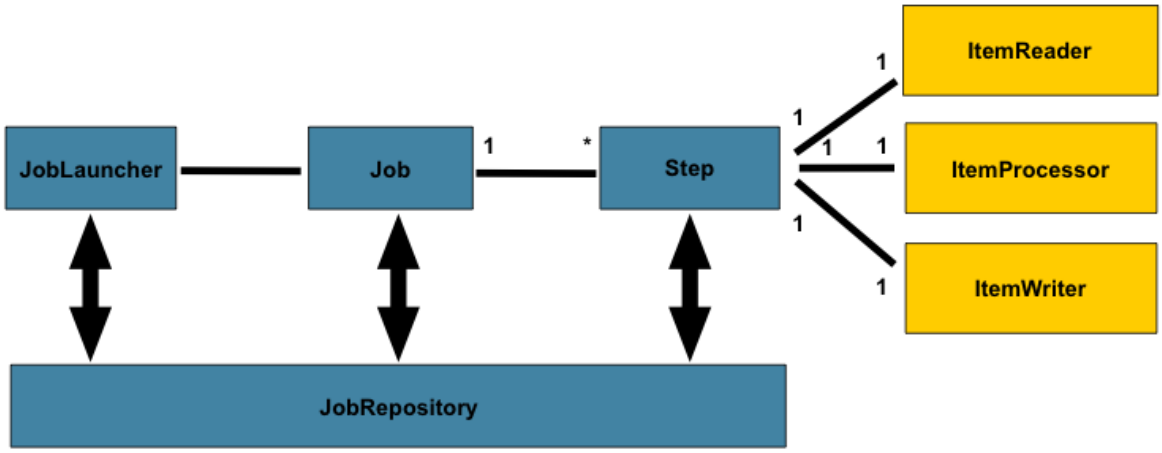
VISIÓN GENERAL

1. **¿Qué es Spring Batch?**

   Spring Batch es un framework ligero enfocado específicamente en la creación de procesos Batch. Provee funciones reutilizables que son esenciales en el procesamiento de grandes volúmenes de registros incluyendo logging/tracing, la gestión de transacciones, las estadísticas de procesamiento de trabajo, el reinicio, la omisión y la gestión de recursos. También proporciona funciones y servicios técnicos mas avanzados que permitirán trabajo por lotes de alto volumen y rendimiento a través de técnicas de optimización y partición.
2. **Casos de uso y aplicaciones prácticas.**

   Un ejemplo de uso es la carga de un fichero enorme con millones de registros; o bien u proceso nocturno que, a partir de una serie de consultas, envía una gran cantidad de e-mails, sms, etc.
3. **Arquitectura de Spring Batch.**

   Componentes de Spring Batch



En el diseño mostrado en la figura podemos observar diferentes elementos:

- **JobRepository:** es el componente encargado de la persistencia de metadatos relativos a los procesos tales como procesos en curso o estados de las ejecuciones.
  Spring Batch está pensando para que la información de los procesamientos quede almacenada en un repositorio persistente o bien en memoria. Este repositorio se utiliza

sobre todo para escritura, aunque también es consultado para comprobar si ya se ha procesado un fichero previamente. También se puede utilizar por si se produce un job fallido, para que en lugar de re-procesar todo el fichero de nuevo, únicamente se re-procese el trozo que ha fallado. El JobRepository escribe y consulta una serie de tablas existentes en base de datos.

- **JobLauncher:** es el componente encargado de lanzar los procesos suministrando los parámetros de entrada deseados. Un Job necesita ser ejecutado por un JobLuncher.
- **Job**: El Job es la representación del proceso. Un proceso, a su vez, es un contenedor de pasos (steps). Es un contenedor de la lógica del proceso por lotes.
- **Step:** Un step (paso) es un elemento independiente dentro de un Job (un proceso) que representa una de las fases de las que está compuesto dicho proceso. Un proceso (Job) debe tener, al menos, un step. Se puede configurar y ejecutar de manera independiente. Un step se compone de tres elementos principales un reader, un writer y un processor donde:
    - ItemReader: Elemento responsable de leer datos de una fuente de datos (BBDD, fichero, cola de mensajes, etc…)
    - ItemProcessor: Elemento responsable tratar la información obtenida por el reader. No es obligatorio su uso.
    - ItemWriter: Elemento responsable guardar la información leída por el reader o tratada por el processor. Si hay un reader debe haber un writer.
- **Tasklet:** Un step no tiene que estar compuesto por un reader, processor y writer. También puede tener únicamente una lógica de negocio. Es el caso del tasklet con el código que se desea ejecutar en el step.
- **Job configuration:** La configuración del Job se realiza utilizando JavaConfig o XML. Esta configuración define los Jobs, Steps, ItemReaders, ItemProcessors, ItemWriters y otros componentes necesarios.
- **Execution Context:** El Execution Context es un almacenamiento de datos persistente que permite compartir datos entre diferentes Steps de un Job. Esto es útil para mantener el estado y los datos intermedios entre las diferentes etapas del procesamiento.
- **Meta-Data Shcema:** Una vez iniciada una aplicación Spring Batch, se establece una conexión con la base de datos que contiene el esquema de tablas que utiliza el framework. Si no existe, se puede incluir por configuración que sea el propio framework el que cree el esquema de base de datos. También disponemos de la opción de cambiar el prefijo del nombre que tendrán estas tablas.

CONFIGURACIÓN DEL ENTORNO

Configuración del proyecto.

Configurar el entorno para usar Spring Batch implica preparar una serie de componentes y dependencias en tu proyecto. Aquí hay una guía paso a paso para configurar un entorno típico para desarrollar y ejecutar trabajos de Spring Batch:

1. **Configurar el Proyecto**

   Dependencias

Primero, debes agregar las dependencias necesarias a tu proyecto. Si estás usando Maven, tu archivo pom.xml debería incluir las siguientes dependencias:

```
<dependencies>
    <!-- Spring Boot Starter Batch -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-batch</artifactId>
    </dependency>

    <!-- Spring Boot Starter JDBC -->
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
    </dependency>

    <!-- H2 Database (para desarrollo y pruebas) -->
    <dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
    </dependency>

    <!-- Otras dependencias necesarias -->
    <!-- Añade tus dependencias específicas, como controladores JDBC para tu base de datos
</dependencies>
```

2. **Configuración de Base de Datos.**

   Spring Batch requiere una base de datos para almacenar los metadatos del trabajo. Para el desarrollo y pruebas, puedes usar H2, una base de datos en memoria. Aquí está la configuración básica en el archivo application.properties.

```
# Configuración de H2 Database
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.datasource.platform=h2

# Mostrar la consola web de H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

   Para un entorno de producción, deberías configurar una base de datos persistente como MySQL, PostgreSQL, etc. Asegúrate de incluir las dependencias correspondientes para el controlador JDBC y configurar los detalles de la conexión en application.properties.

```
# Configuración de MySQL Database
spring.datasource.url=jdbc:mysql://localhost:3306/springbatch
spring.datasource.username=root
spring.datasource.password=password
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.platform=mysql
```

3. **Definir los Beans de Spring Batch**

   Job Repository

   El JobRepository se configura automáticamente por Spring Boot si se habilita @EnableBatchProcessing. Puedes personalizarlo si es necesario.

```java
@Configuration
public class BatchConfig {

    @Autowired
    private DataSource dataSource;

    @Autowired
    private PlatformTransactionManager transactionManager;

    @Bean
    public JobRepository jobRepository() throws Exception {
        JobRepositoryFactoryBean factory = new JobRepositoryFactoryBean();
        factory.setDataSource(dataSource);
        factory.setTransactionManager(transactionManager);
        factory.setDatabaseType(DatabaseType.H2.getProductName());
        return factory.getObject();
    }

    @Bean
    public JobLauncher jobLauncher() throws Exception {
        SimpleJobLauncher jobLauncher = new SimpleJobLauncher();
        jobLauncher.setJobRepository(jobRepository());
        return jobLauncher;
    }

    @Bean
    public JobExplorer jobExplorer() throws Exception {
        JobExplorerFactoryBean jobExplorerFactoryBean = new JobExplorerFactoryBean();
        jobExplorerFactoryBean.setDataSource(dataSource);
        return jobExplorerFactoryBean.getObject();
    }
}
```

4. Definir Jobs y Steps

```java
@Configuration
public class BatchConfiguration {

    @Autowired
    private JobBuilderFactory jobBuilderFactory;

    @Autowired
    private StepBuilderFactory stepBuilderFactory;

    @Bean
    public Job importUserJob(JobCompletionNotificationListener listener, Step step1) {
        return jobBuilderFactory.get("importUserJob")
            .incrementer(new RunIdIncrementer())
            .listener(listener)
            .flow(step1)
            .end()
            .build();
    }

    @Bean
    public Step step1(ItemReader<User> reader, ItemProcessor<User, User> processor, ItemWr
        return stepBuilderFactory.get("step1")
            .<User, User> chunk(10)
            .reader(reader)
            .processor(processor)
            .writer(writer)
            .build();
    }

    @Bean
    public ItemReader<User> reader() {
        return new FlatFileItemReaderBuilder<User>()
            .name("userItemReader")
            .resource(new ClassPathResource("sample-data.csv"))
            .delimited()
            .names(new String[]{"firstName", "lastName"})
            .fieldSetMapper(new BeanWrapperFieldSetMapper<User>() {{
                setTargetType(User.class);
            }})
            .build();
    }

    @Bean
    public ItemProcessor<User, User> processor() {
        return new UserItemProcessor();
    }

    @Bean
    public ItemWriter<User> writer(DataSource dataSource) {
        return new JdbcBatchItemWriterBuilder<User>()
            .itemSqlParameterSourceProvider(new BeanPropertyItemSqlParameterSourceProvider
            .sql("INSERT INTO people (first_name, last_name) VALUES (:firstName, :lastName
            .dataSource(dataSource)
            .build();
    }
}
```

5. Ejecutar el Job

   Puedes ejecutar el trabajo desde la línea de comandos o programando.

INYECCIÓN DE DEPENDENCIAS

En el siguiente ejemplo se busca demostrar la inyección de dependencias, y el desacoplamiento.

Para esto se crearon 4 clases (Jugador, Consola, PC, Main) y una interfaz (Jugable), en un solo paquete.

1. Se crea la clase Jugador que tendrá como atributos el nombre del jugador y un dispositivo para guardar el dispositivo con el que esta jugando el jugador.

Jugador.java | Consola.java | Pc.java | Main.java

```java
package com.academiajava;

public class Jugador {
    private String nombre;
    private Jugable dispositivoDeJuego;

    public Jugador(String nombre, Jugable dispositivoDeJuego) {
        this.nombre = nombre;
        this.dispositivoDeJuego = dispositivoDeJuego;
    }

    public void jugar() {
        this.dispositivoDeJuego.jugar();
    }

    public String getNombre() {
        return nombre;
    }

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

2. Se crea la interfaz Jugable que establecerá el método jugar característico de todos los dispositivos.

Jugable.java | Jugador.java | Consola.java

```java
package com.academiajava;

public interface Jugable {

    void jugar();

}
```

3. Se crean las clases de consola y PC que implementaran Jugable, las instancias de los dispositivos requieren distinguirse por su modelo, por lo que se establece como atributo, en su constructor se indicará el modelo.

Pc.java

```java
package com.academiajava;

public class Pc implements Jugable{
    private String modelo;

    public Pc(String modelo) {
        this.modelo = modelo;
    }

    @Override
    public void jugar() {
        System.out.println("Estoy jugando con la PC: "+modelo);
    }

}
```

Consola.java | Jugable.java | Jugador.java | Main.java

```java
package com.academiajava;

public class Consola implements Jugable{
    private String modelo;

    public Consola(String modelo) {
        this.modelo = modelo;
    }

    @Override
    public void jugar() {
        System.out.println("Estoy jugando con la consola: "+modelo);
    }

}
```

4. Se crea la clase Main para demostrar la inyección de dependencias, en esta clase se observa la creación de dos instancias de Jugable, lo que indica es que el jugador puede hacer uso o de una PC o de una consola sin tener dependencia al dispositivo, es decir la creación de un jugador no genera dependencia con su atributo por lo que puede haber nuevos dispositivos para jugar sin afectar el código.

Main.java

```java
package com.academiajava;

public class Main {

    public static void main(String[] args) {
        Jugable dispositivo1 = new Pc("Dell");
        Jugable dispositivo2 = new Consola("PS4");

        Jugador jugador =  new Jugador("Dan", dispositivo2);
        jugador.jugar();
    }

}

```