Java

Tipi di dato astratto/information hiding

I tipi di dato astratto sono utilizzati per nascondere i dettagli implementativi e fornire un'interfaccia chiara. Ecco un esempio di information hiding con una classe in Java:

```java
public class Account {
    private double balance;

    public Account(double initialBalance) {
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        // Implementazione deposito
    }

    public void withdraw(double amount) {
        // Implementazione prelievo
    }

    public double getBalance() {
        return balance;
    }
}
```

Classi, oggetti, attributi, metodi in diagrammi UML

In un diagramma UML, le classi sono rappresentate da rettangoli con tre sezioni: nome della classe, attributi e metodi. Ecco un esempio per la classe `Person`:

```
+----------------------+
|      Person          |
+----------------------+
| - name: String       |
| - age: int           |
+----------------------+
```

```
| + getName(): String  |
| + getAge(): int      |
| + setName(name: String): void |
| + setAge(age: int): void |
+----------------------+
```

Ereditarietà/polimorfismo

Ereditarietà e polimorfismo sono concetti fondamentali in Java. Ecco un esempio di ereditarietà:

```java
public class Shape {
    // Metodi e attributi comuni a tutte le forme
}

public class Circle extends Shape {
    // Metodi e attributi specifici per un cerchio
}

public class Square extends Shape {
    // Metodi e attributi specifici per un quadrato
}
```

Associazioni tra classi

Le associazioni tra classi indicano le relazioni tra gli oggetti. Ad esempio, una relazione uno-a-molti tra `Author` e `Book`:

```
+---------------------+       +------------------+
|       Author        |       |       Book       |
+---------------------+       +------------------+
| - name: String      |       | - title: String  |
| - books: List<Book> |       | - author: Author |
+---------------------+       +------------------+
```

Convenzioni di codifica del linguaggio Java

Le convenzioni di codifica Java includono l'uso di CamelCase per i nomi di variabili e metodi, inizializzatori statici, indentazione e altro. Ad esempio:

```java
public class ExampleClass {
    private int exampleVariable;

    public void setExampleVariable(int value) {
        exampleVariable = value;
    }

    public int getExampleVariable() {
        return exampleVariable;
    }
}
```

Record

I record sono una nuova funzionalità in Java per la creazione di classi immutabili con sintassi abbreviata. Ecco un esempio:

```java
public record Point(int x, int y) {
    // Nessun metodo aggiuntivo, automaticamente generato dal compilatore
}
```

Tipi di dato primitivi, enumerativi, classi wrapper

Java ha tipi di dato primitivi come int, float, char, ecc. Gli enumerativi sono dichiarati con la keyword `enum`. Le classi wrapper forniscono oggetti per i tipi primitivi, come `Integer` per `int`.

```java
int primitiveType = 42;

enum Days { MONDAY, TUESDAY, WEDNESDAY }

Integer wrapperType = 42;
```

Stringhe di caratteri e codifica Unicode

Le stringhe in Java sono rappresentate dalla classe `String`. La codifica di caratteri predefinita è UTF-16. Esempio:

```java
String message = "Ciao, mondo!";
```

Gestione date ed orari

La gestione di date e orari in Java è fatta tramite le classi del package `java.time`. Ecco un esempio:

```java
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.LocalDateTime;

public class DateTimeExample {
    public static void main(String[] args) {
        LocalDate currentDate = LocalDate.now();
        LocalTime currentTime = LocalTime.now();
        LocalDateTime currentDateTime = LocalDateTime.now();

        System.out.println("Data: " + currentDate);
        System.out.println("Ora: " + currentTime);
        System.out.println("Data e Ora: " + currentDateTime);
    }
}
```

Documentazione dei programmi con JavaDoc

JavaDoc è uno strumento per generare documentazione automatica. Ecco un esempio:

```java
/**
 * Questa classe rappresenta un esempio di documentazione JavaDoc.
 */
public class ExampleClass {
    /**
     * Questo metodo esegue un'operazione specifica.
     * @param value Il valore su cui eseguire l'operazione.
     * @return Il risultato dell'operazione.
     */
    public int performOperation(int value) {
        // Implementazione
        return result;
    }
}
```

Array in Java

Gli array in Java sono oggetti che contengono elementi di uno stesso tipo. Esempio:

```
int[] array = {1, 2, 3, 4, 5};
```

Oggetti e riferimenti in Java

In Java, le variabili degli oggetti contengono riferimenti agli oggetti. Esempio:

```
Person person = new Person("Mario", 25);
```

Array come parametri e valori restituiti dai metodi di una classe

Gli array possono essere passati come parametri e restituiti dai metodi. Esempio:

```
public class ArrayExample {
    public void modifyArray(int[] arr) {
        // Modifica l'array
    }

    public int[] createArray() {
        // Crea e restituisce un nuovo array
    }
}
```

Spero che questo ti fornisca una panoramica chiara di questi concetti in Java! Se hai domande o hai bisogno di ulteriori chiarimenti su uno specifico argomento, non esitare a chiedere.

Eccezioni

Le eccezioni in Java sono gestite attraverso i blocchi `try`, `catch`, e `finally`. Ecco un esempio di come gestire un'eccezione:

```
try {
    // Codice che potrebbe generare un'eccezione
    int result = 10 / 0;
} catch (ArithmeticException e) {
    // Gestione dell'eccezione
```

```
    System.out.println("Errore: divisione per zero");
} finally {
    // Blocco eseguito sempre, indipendentemente dalla presenza di un'eccezione
    System.out.println("Questo blocco viene sempre eseguito");
}
```

Gestione I/O predefinito

Per la gestione dell'I/O standard, puoi utilizzare le classi `System.in`, `System.out`, e `System.err`. Ecco un esempio di lettura da input standard:

```
import java.util.Scanner;

public class InputExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Inserisci una frase: ");
        String input = scanner.nextLine();
        System.out.println("Hai inserito: " + input);
    }
}
```

Gestione I/O file di testo

Per la gestione di file di testo, puoi utilizzare le classi `File`, `FileReader`, e `BufferedReader`. Ecco un esempio di lettura da un file di testo:

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

public class FileReadExample {
    public static void main(String[] args) throws IOException {
        File file = new File("nomefile.txt");
        BufferedReader reader = new BufferedReader(new FileReader(file));

        String line;
        while ((line = reader.readLine()) != null) {
            System.out.println(line);
        }
```

```
        reader.close();
    }
}
```

Serializzazione e persistenza

La serializzazione in Java consente di convertire oggetti in un formato che può essere salvato su un file o trasmesso attraverso la rete. Ecco un esempio:

```
import java.io.*;

class Person implements Serializable {
    String name;
    int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }
}

public class SerializationExample {
    public static void main(String[] args) throws IOException,
ClassNotFoundException {
        Person person = new Person("Mario", 25);

        // Serializzazione
        ObjectOutputStream objectOutputStream = new ObjectOutputStream(new
FileOutputStream("person.ser"));
        objectOutputStream.writeObject(person);
        objectOutputStream.close();

        // Deserializzazione
        ObjectInputStream objectInputStream = new ObjectInputStream(new
FileInputStream("person.ser"));
        Person deserializedPerson = (Person) objectInputStream.readObject();
        objectInputStream.close();

        System.out.println("Nome: " + deserializedPerson.name + ", Età: " +
deserializedPerson.age);
```

```
    }
}
```

Strutture dati

Implementazione di una lista

Puoi utilizzare la classe `ArrayList` per implementare una lista dinamica in Java:

```
import java.util.ArrayList;

public class ListExample {
    public static void main(String[] args) {
        ArrayList<String> lista = new ArrayList<>();
        lista.add("Elemento 1");
        lista.add("Elemento 2");
        lista.add("Elemento 3");

        System.out.println("Contenuto della lista: " + lista);
    }
}
```

Il pattern Iterator

L'iteratore è utilizzato per attraversare una collezione senza esporre i dettagli interni della struttura dati. Ecco un esempio con `ArrayList`:

```
import java.util.ArrayList;
import java.util.Iterator;

public class IteratorExample {
    public static void main(String[] args) {
        ArrayList<String> lista = new ArrayList<>();
        lista.add("Elemento 1");
        lista.add("Elemento 2");
        lista.add("Elemento 3");

        Iterator<String> iterator = lista.iterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
```

```
    }
}
```

Le classi nidificate

Le classi nidificate sono classi definite all'interno di un'altra classe. Ecco un esempio:

```
public class OuterClass {
    private int outerVariable;

    public class InnerClass {
        public void display() {
            System.out.println("Valore variabile esterna: " + outerVariable);
        }
    }

    public static void main(String[] args) {
        OuterClass outer = new OuterClass();
        OuterClass.InnerClass inner = outer.new InnerClass();
        inner.display();
    }
}
```

La pila e la coda

Puoi utilizzare le classi `Stack` e `LinkedList` per implementare una pila e una coda, rispettivamente:

```
import java.util.Stack;
import java.util.LinkedList;
import java.util.Queue;

public class StackQueueExample {
    public static void main(String[] args) {
        // Pila
        Stack<String> stack = new Stack<>();
        stack.push("Elemento 1");
        stack.push("Elemento 2");
        stack.push("Elemento 3");

        System.out.println("Contenuto della pila: " + stack);
```

```
        // Coda
        Queue<String> queue = new LinkedList<>();
        queue.add("Elemento 1");
        queue.add("Elemento 2");
        queue.add("Elemento 3");

        System.out.println("Contenuto della coda: " + queue);
    }
}
```