Java vs Rust – Main Differences with Examples

This document compares **Java** and **Rust** focusing on syntax, concepts, and common usage patterns, with side-by-side examples.

---

1. Console Output

Java

```java
System.out.println("Hello, world");
```

Rust

```rust
println!("Hello, world");
```

Key differences - Java uses methods on `System.out` - Rust uses macros (`println!`) for formatted output

---

2. Variables and Constants Initialization

Java (most used data types)

```java
int a = 10;
double b = 3.14;
boolean flag = true;
char c = 'A';
String s = "Hello";

final int CONST_VAL = 100;
```

Rust (most used data types)

```rust
let a: i32 = 10;
let b: f64 = 3.14;
let flag: bool = true;
let c: char = 'A';
let s: String = String::from("Hello");

const CONST_VAL: i32 = 100;
```

Key differences - Java variables are mutable by default - Rust variables are **immutable by default** (`let mut x` for mutability) - Rust constants must have explicit types

3. Conditionals

Java

```java
if (a > 10) {
    System.out.println("Greater");
} else if (a == 10) {
    System.out.println("Equal");
} else {
    System.out.println("Smaller");
}
```

Rust

```rust
if a > 10 {
    println!("Greater");
} else if a == 10 {
    println!("Equal");
} else {
    println!("Smaller");
}
```

Key differences - Rust does not require parentheses around conditions - `if` in Rust is an expression and can return values

---

4. Loops

Java

```java
for (int i = 0; i < 5; i++) {
    System.out.println(i);
}

int i = 0;
while (i < 5) {
    i++;
}
```

Rust

```rust
for i in 0..5 {
    println!("{}", i);
}
```

```rust
let mut i = 0;
while i < 5 {
    i += 1;
}
```

---

5. Arrays

Java

```java
int[] arr = {1, 2, 3};
System.out.println(arr[0]);
```

Rust

```rust
let arr: [i32; 3] = [1, 2, 3];
println!("{}", arr[0]);
```

Key differences - Rust arrays have fixed size known at compile time

---

6. Lists / Dynamic Collections

Java (List)

```java
List<Integer> list = new ArrayList<>();
list.add(1);
list.add(2);
```

Rust (Vector)

```rust
let mut list: Vec<i32> = Vec::new();
list.push(1);
list.push(2);
```

---

7. Classes vs Structs

Java Class

```java
class Person {
    String name;
    int age;
```

```
    void greet() {
        System.out.println("Hi, I'm " + name);
    }
}
```

Rust Struct + Impl

```
struct Person {
    name: String,
    age: u32,
}

impl Person {
    fn greet(&self) {
        println!("Hi, I'm {}", self.name);
    }
}
```

---

8. Object Initialization

Java

```
Person p = new Person();
p.name = "Alice";
p.age = 30;
```

Rust

```
let p = Person {
    name: String::from("Alice"),
    age: 30,
};
```

---

9. Inheritance

Java (Class Inheritance)

```
class Animal {
    void speak() {
        System.out.println("Animal sound");
    }
}
```

```java
class Dog extends Animal {
    @Override
    void speak() {
        System.out.println("Bark");
    }
}
```

Rust (Traits – No Classical Inheritance)

```rust
trait Animal {
    fn speak(&self);
}

struct Dog;

impl Animal for Dog {
    fn speak(&self) {
        println!("Bark");
    }
}
```

Key differences - Java uses class inheritance - Rust uses **traits** (composition over inheritance)

---

10. Polymorphism – Strategy Design Pattern

Java Strategy Example

```java
interface PaymentStrategy {
    void pay(int amount);
}

class CreditCardPayment implements PaymentStrategy {
    public void pay(int amount) {
        System.out.println("Paid " + amount + " with credit card");
    }
}

class PaymentContext {
    private PaymentStrategy strategy;

    public PaymentContext(PaymentStrategy strategy) {
        this.strategy = strategy;
    }

    public void execute(int amount) {
        strategy.pay(amount);
```

```
    }
}
```

Rust Strategy Example

```rust
trait PaymentStrategy {
    fn pay(&self, amount: i32);
}

struct CreditCardPayment;

impl PaymentStrategy for CreditCardPayment {
    fn pay(&self, amount: i32) {
        println!("Paid {} with credit card", amount);
    }
}

struct PaymentContext {
    strategy: Box<dyn PaymentStrategy>,
}

impl PaymentContext {
    fn execute(&self, amount: i32) {
        self.strategy.pay(amount);
    }
}
```

Key differences - Java uses interfaces + dynamic dispatch automatically - Rust uses `dyn Trait` and explicit heap allocation (`Box`)

---

11. Database Management

Java – JDBC Example

```
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/test", "user", "pass"
);

PreparedStatement stmt = conn.prepareStatement(
    "SELECT * FROM users"
);
ResultSet rs = stmt.executeQuery();
```

Rust – Equivalent (SQLx Example)

```rust
use sqlx::mysql::MySqlPool;

let pool = MySqlPool::connect("mysql://user:pass@localhost/test").await?;

let rows = sqlx::query!("SELECT * FROM users")
    .fetch_all(&pool)
    .await?;
```

Key differences - Java JDBC is blocking by default - Rust database libraries are usually **async** - Rust emphasizes compile-time query checking (SQLx)

---

12. Database Table to Object Mapping Example (Client)

This example shows how to read data from a **Client** table and map each row into a **Client** object, storing them in a list.

Database Table: Client

Column	Type
id	INT
name	VARCHAR
surname	VARCHAR
birthdate	DATE
phone_number	VARCHAR

Java Example (JDBC)

Client Class

```java
import java.time.LocalDate;

class Client {
    int id;
    String name;
    String surname;
    LocalDate birthdate;
    String phoneNumber;

    public Client(int id, String name, String surname, LocalDate birthdate,
String phoneNumber) {
        this.id = id;
```

```
        this.name = name;
        this.surname = surname;
        this.birthdate = birthdate;
        this.phoneNumber = phoneNumber;
    }
}
```

Program Logic

```
List<Client> clients = new ArrayList<>();

Connection conn = DriverManager.getConnection(
    "jdbc:mysql://localhost:3306/test", "user", "pass"
);

PreparedStatement stmt = conn.prepareStatement(
    "SELECT id, name, surname, birthdate, phone_number FROM client"
);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {
    Client client = new Client(
        rs.getInt("id"),
        rs.getString("name"),
        rs.getString("surname"),
        rs.getDate("birthdate").toLocalDate(),
        rs.getString("phone_number")
    );
    clients.add(client);
}
```

Rust Example (SQLx)

Client Struct

```
use chrono::NaiveDate;

struct Client {
    id: i32,
    name: String,
    surname: String,
    birthdate: NaiveDate,
    phone_number: String,
}
```

Program Logic

```rust
use sqlx::mysql::MySqlPool;

let pool = MySqlPool::connect("mysql://user:pass@localhost/test").await?;

let rows = sqlx::query!(
    "SELECT id, name, surname, birthdate, phone_number FROM client"
)
.fetch_all(&pool)
.await?;

let mut clients: Vec<Client> = Vec::new();

for row in rows {
    let client = Client {
        id: row.id,
        name: row.name,
        surname: row.surname,
        birthdate: row.birthdate,
        phone_number: row.phone_number,
    };
    clients.push(client);
}
```

Key differences - Java uses a mutable `ResultSet` and manual field extraction - Rust maps query results into strongly typed fields at compile time (SQLx) - Rust enforces ownership and immutability guarantees on the `Client` data

---

13. Summary Table

Feature	Java	Rust
Memory management	Garbage Collector	Ownership & Borrowing
Mutability	Mutable by default	Immutable by default
Inheritance	Class-based	Trait-based
Polymorphism	Interfaces	Traits + `dyn`
Concurrency	Threads	Fearless concurrency
Database access	JDBC	Async drivers (SQLx, Diesel)

Conclusion: Java prioritizes productivity and runtime flexibility, while Rust prioritizes safety, performance, and explicit control. Both can model the same designs (like Strategy), but Rust makes ownership and dispatch choices explicit.