20CYS312 - Principles of Programming Languages
Exploring Programming Paradigms

Assignment-01

Presented by Yaswanth G
CB.EN.U4CYS21089
TIFAC-CORE in Cyber Security
Amrita Vishwa Vidyapeetham, Coimbatore Campus

Feb 2024

AMRITA
VISHWA VIDYAPEETHAM

Outline

Aspect-Oriented Paradigm

Aspect-Oriented Programming (AOP) is a technique that helps keep your code smart, neat, and organized. So, instead of mixing up these extra tasks with the main job your code is supposed to do (which can make it messy), AOP lets you add these tasks separately. It's like telling the program, "Hey, whenever you see something specific happening in the code (we call it a 'pointcut'), do this extra thing (we call it 'advice') without changing the original code."

1. modularity
2. Aspect
3. separation of cross-cutting concern
4. Advice
5. Pointcut
6. joinpoint

AOP Example Explaination

1. void transfer (Account fromAcc , Account toAcc , i n t amount) throws Exception if (fromAcc.getBala n c e ( ) < amount) throw new I n s uffi ci e n t F u n d sE x c e p ti o n ( ) ; fromAcc . withdraw (amount) ; toAcc . d e p o si t (amount) ;
2. void t r a n s f e r (Account fromAcc , Account toAcc , i n t amount , User u se r , Logger logg e r , Database da taba se) throws Excep tion l o g g e r . i nf o (" T r a n sf e r ri n g money . . . ") ; i f (! i s U s e r A u t h o ri s e d (u se r , fromAcc)) l o g g e r . i nf o (" User has no p e rmi s sio n . ") ; throw new Unau tho ri sedU se rE xcep tion ( ) ; i f (fromAcc . g e tBala n c e ( ) < amount) l o g g e r . i nf o (" I n s u f f i c i e n t fund s . ") ; throw new I n s uffi ci e n t F u n d sE x c e p ti o n ( ) ; 2 | 20CYS312 - Principles of Programming Languages fromAcc . withdraw (amount) ; toAcc . d e p o si t (amount) ; da taba se . commitChanges ( ) ; // Atomic o p e r a ti o n . l o g g e r . i nf o (" T ra n sa c tio n s u c c e s s f u l . ") ;

JBoss

1. Implementing Aspect Class
2. Advice
3. Pointcut and Joinpoint

Imperative

1. Imperative programming is a development approach that allows for modifying a program's state through the use of statements, which are syntactic units in an imperative programming language.
2. In Imperative programming, the focus is on providing a sequence of statements that explicitly instruct the computer on how to perform a task or achieve a particular result.

Rust

1. type safety
2. Concurrency
3. Memory safety
4. Reliability
5. Immutability
6. Higher-Order functions
7. Error Handling

Rust Example

```
// D efi n
e a s t r u c t f o r a bank accoun t s t r u c t BankAccount  account_number : u32, balance :
f64, implBankAccount//Constructortocreateanewbankaccountfnnew(account_number : u32,
: expr , account : expr) => println!(""",message ,
account);; fnmain()//Createtwobankaccountsletmutaccount1 = BankAccount :: new(12345
```

Comparison - AOP and Imperative

Aspect-Oriented Paradigm (AOP)	Imperative Paradigm
Both involve specifying step-by-step instructions for the computer to follow.	Similar in the sense that both paradigms rely on imperative instructions to dictate program flow.
Both aim to enhance code modularity by organizing code into manageable units.	Achieves code modularity through functions, procedures, and modules, promoting reusability and maintainability.
Applied in real-world scenarios, addressing complex programming requirements.	Imperative paradigm is found extensive use in various real-world applications, demonstrating versatility.

Aspect-Oriented Paradigm (AOP)	Imperative Paradigm
AOP addresses concerns using aspects, allowing for the separation of concerns that cut across different modules.	Concerns are encapsulated within functions or modules, and explicit separation is achieved through traditional modularization.
AOP dynamically alters the execution flow at runtime, providing flexibility in managing cross-cutting concerns.	Imperative languages follow a static, sequential execution flow, with control structures determining the order of operations.
AOP specifically focuses on handling cross-cutting concerns, which affect multiple modules or aspects of a program.	Cross-cutting concerns may be addressed using modularization, but the approach is more explicit and manual compared to AOP.

Comparison - JBoss and Rust

Similarities between JBoss and Rust:

1. Object-Oriented Paradigm
2. Concurrency Support
3. Modularity and Reusability
4. Memory Safety

Distinctions between JBoss AOP and Rust Imperative:

1. Applications and Subjects Memory Management
2. Systems programming vs. aspect-oriented programming

References

1. https://www.javatpoint.com/what-is-imperative-programming
2. https://en.wikipedia.org/wiki/Imperative_programming
3. https://www.researchgate.net/publication/320673013_ActionPool_A_NOVEL_DYNA
4. https://doc.rust-lang.org/book/foreword.html
5. https://en.wikipedia.org/wiki/Rust_(programming_language)
6. https://en.wikipedia.org/wiki/Aspect-oriented_programming
7. https://romain-b.medium.com/pros-and-cons-of-imperative-and-functional-programming-paradigms-to-solve-the-same-technical-1511ac2f654c
8. https://chat.openai.com/chat