

introduction to

JAVA

manual book for beginners

By
Tin Mai Zaw





အမှာစကား

ဤစာအုပ်သည် Java Programming ကို အခြေခံမှစတင် လေ့လာမည့် သူများအတွက် လက်စွဲစာအုပ်ဖြစ်ပါတယ်။ ဒုတိယအကြိမ် မြန်မာလို ထုတ်ဝေခြင်းဖြစ်ပါတယ်။ Java Reference Books, Online Reference Website တွေအပြင် Chat GPT, Deepseek AI tools တွေထံးပြီး အချိုသင်ခန်းစာတွေကို ထပ်မံ ဖြည့်စွက်ထားပါတယ်။ ဂွန်ပျိုးတာကျောင်း သူကျောင်းသားများ၊ အင်ဂျင်နီယာအိုင်တီ ကျောင်းသူကျောင်းသားများကို တစ်ဖက်တစ်လမ်း အထောက်အကူ ပြုစေရန်အတွက် ရည်ရွယ် ထုတ်ဝေရခြင်းလည်းဖြစ်ပါတယ်။ အိုင်တီနဲ့ အသက်မွေး ဝမ်းကြောင်းပြုလိုသူများ၊ software programmer ဖြစ်ချင်သူများ၊ ကမ္ဘာကျော် software application တွေတိတွင်ပြီး millionaire သူငြေားဖြစ်ဖို့ စိတ်ကူးအိမ်မက်ရှိသူ မည်သူမဆို လွယ်လွယ်ကူကူ လေ့လာနိုင်အောင် လေ့ကျင့်ခန်း ဥပမာများ၊ လက်တွေ့ real life application သင်ခန်းစာ များနဲ့ တွဲပြီးတော့ ပြည့်ပြည့်စုစုပေါင်း ရေးသားဖန်းထားတဲ့ စာအုပ်ဖြစ်ပါတယ်။

Programming ကို လုံးဝမထိတွေ့သေးတဲ့သူတွေကတော့ step by step လေ့လာသွားရပါမယ်။ Programming experience နည်းနည်းပါးပါး ရှိတဲ့သူတွေကတော့ methods ကနေ စပြီး လေ့လာဖို့ အကြံပြုပါတယ်။ Method ကို နားလည်မှ OOP ကို ကောင်းစွာ နားလည်လာမှာ မို့ပါ။ OOP ကို ကောင်းကောင်းနားလည် သဘောပေါက်ထားရင် software development architecture တွေဖြစ်တဲ့ Model-Repository-Services Architecture, Model-View-Controller (MVC) Architecture နဲ့ Model-View-Template (MVT) တွေကို အလွယ်တကူ လေ့လာနိုင်မယ်။ Quality ရှိတဲ့ Software Production တွေကို လုပ်လာနိုင်မှာ ဖြစ်ပါတယ်။

OOP ကို လေ့လာပြီးသွားရင် Mini Project လေးတွေကို တစ်ခုပြီးတစ်ခု လုပ်ရပါမယ်။ ဒါ စာအုပ်ထဲမှာတော့ Mini Project တွေကို Model-Repository-Services ပုံစံဝအောင်ရေးပေးထားပါတယ်။ Mini Project application တွေရေးတတ်သွားပြီဆို Mini Game application တွေကို ဆက်ပြီးလေ့ကျင့်ရမှာပါ။ Game code တွေကို အချိန်ပိုပေးပြီး study လုပ်ရပါမယ်။ Game application တွေရေးရင် code challenges တွေပျေားပါတယ်။ အချိန်ပိုပေးရပါတယ်။ Game မရေးခင်မှာ game rules တွေကို အင် သိရပါမယ်။ Logical complex တွေကြောင့် study လုပ်တဲ့အခါ ပိုပြီးအားစိုက်ရပါတယ်။ ဒါစာအုပ်ထဲက game project တွေအပြင် အခြား ကိုယ်သိတဲ့ game application တွေကို လည်း လေ့ကျင့်ရေးသားဖို့ တိုက်တွေးပါတယ်။

Coding Experience နည်းနည်းရှိသွားပြီဆိုရင်တော့ Algorithms တွေ နဲ့ Software Development Architecture တွေကို စလေ့လာလိုရပါပြီ။ Study လုပ်တဲ့အခါ code တွေကို copy and paste မလုပ်ပဲ manual code typing လုပ်ပြီးမှ လေ့လာသင့်ပါတယ်။ ကြည့်ပြီး ခါ ခါ လောက ရေးပြီးမှ အလွတ် ပြန်ရေးကျင့်ဖို့ တိုက်တွေးပါတယ်။ Code တွေနဲ့ ရင်းနှီးမှ သာ logic တွေ algorithms တွေကို ပိုပြီးနားလည် လာပါလိမ့်မယ်။



မာတိကာ

1. History	Page	4
2. Why Java?	Page	6
3. Eclipse Installation	Page	9
4. Data Types	Page	18
5. Variables	Page	22
6. Basic Calculations	Page	26
7. Conditional Statements	Page	30
8. Loops	Page	40
9. String Manipulations	Page	46
10. Static Array	Page	50
11. Methods	Page	57
12. OOP	Page	63
13. Java Collections	Page	83
14. List Interface	Page	87
15. Set Interface	Page	91
16. Map Interface	Page	95
17. Lambda	Page	99
18. Date	Page	106
19. Regular Expression	Page	110
20. Random	Page	115
21. File Handling	Page	121



22. Serialization	Page	137
23. Exception Handling	Page	153
24. Testing and Debugging	Page	158
25. Thread	Page	164
26. Socket	Page	172
27. JDBC	Page	179
28. Mini Projects	Page	202
29. Mini Games	Page	289
30. Algorithms	Page	304
31. Software Development Architectures	Page	324
32. Best Practices	Page	329
33. Oracle Certifications	Page	335
34. Online References for Self-Study	Page	338
35. References	Page	341
36. ဘရေးသူ ကိုယ်ရေးအကျဉ်း	Page	342
37. ဘရေးသူ၏ အိုင်တီအတွေ့အကြံမှတ်တမ်း	Page	343
38. Documentary Photos	Page	344



HISTORY



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



JAVA HISTORY

Java ကို 1991 ခုနှစ်မှာ Sun Microsystems က James Gosling နဲ့ အဖွဲ့က "Green Project" အဖြစ် စတင်ခဲ့ပါတယ်။ အဲဒီတုန်းက အမည်က "Oak" လို့ခေါ် ပြီး အိမ်သုံးပစ္စည်းတွေအတွက် programming language အဖြစ် ရည်ရွယ်ခဲ့ပါတယ်။ ဒါပေမယ့် Internet ဆောင်စာတော်အမှု မှာ Java ကို web development အတွက် ပြောင်းလဲအသုံးပြုဖို့ ဆုံးဖြတ်ခဲ့ပါတယ်။

1995 ခုနှစ်မှာ Java 1.0 ကို ပထမဆုံးအကြိမ် official ထုတ်ပြန်ခဲ့ပါတယ်။ Java ရဲ့ အဓိက အားသာချက်က "Write Once, Run Anywhere" (WORA) principle ဖြစ်ပြီး JVM (Java Virtual Machine) ပေါ်မှာ run နိုင်တဲ့ အတွက် platform-independent ဖြစ်ပါတယ်။ ဒါ feature ကြောင့် Java ကဲ enterprise software development မှာ အရမ်းရေပန်းစားလာခဲ့ပါတယ်။

2000 ခုနှစ်တွေမှာ Java ကို J2SE (Standard Edition), J2EE (Enterprise Edition) နဲ့ J2ME (Micro Edition) ဆိုပြီး သုံးချိုးခွဲခဲ့ပါတယ်။ 2006 မှာ Sun Microsystems က Java ကို open-source အဖြစ် release လုပ်ခဲ့ပြီး OpenJDK project ကို စတင်ခဲ့ပါတယ်။ 2010 မှာ Oracle ကဲ Sun Microsystems ကို ဝယ်ယူပြီးနောက် Java ရဲ့ ဖွံ့ဖြိုးမှုကို ဆက်လက်၍ ဆောင်ခဲ့ပါတယ်။

Java 8 (2014) က major release တစ်ခုဖြစ်ပြီး Lambda Expressions, Stream API, Optional class တို့ငါး modern features တွေ မိတ်ဆက်ပေးခဲ့ပါတယ်။ 2018 မှာ Java ကို 6 လတစ်ကြိမ် update လုပ်ဖို့ new release model ကို စတင်ခဲ့ပါတယ်။ အခုအခါမှာ Java 21 (LTS version) အထိ ရောက်ရှုလှပြုဖြစ်ကာ virtual threads, pattern matching နဲ့ record classes တို့ငါး advanced features တွေ ပါဝင်လာပါတယ်။

နှစ် 30 နှစ်ပါးကြောမြင့်လာတဲ့ Java ဟာ Android development, enterprise applications, cloud computing နဲ့ big data technologies တွေမှာ အဓိက အသုံးပြုနေဆဲဖြစ်ပါတယ်။ လက်ရှိအချိန်မှာလည်း TIOBE Index အရ ကမ္ဘာအသုံးအများဆုံး programming languages တွေထဲက တစ်ခုအဖြစ် ရပ်တည်နေဆဲဖြစ်ပါတယ်။



WHY JAVA?



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



WHY JAVA?

Java Programming Language ကို ကမ္ဘာတစ်ခုမှာ အသုံးများရတဲ့ အဓိက အကြောင်းရင်း တွေကတော့ Platform Independent ဖြစ်လိုပါ။ ဘယ် OS မှာ မဆို အလုပ်လုပ်နိုင်ပါတယ်။ "Write Once, Run Anywhere" ဆိုတဲ့ အားသာချက်ရှိပါတယ်။ JVM (Java Virtual Machine) ကြောင့် Windows, Mac, Linux စသည့် OS အားလုံးမှာ run နိုင်ပါတယ်။

Object-Oriented Programming (OOP) ကို အပြည့်အဝ ထောက်ပံ့နိုင်တဲ့ အတွက် ကြီးမားတဲ့ large-scale applications, နိုင်ငံတော် project တွေနဲ့ အထူးသင့်တော်တဲ့ language ဖြစ်ပါတယ်။

အားကောင်းတဲ့ Memory Management တွေကို လုပ်နိုင်ပါတယ်။ Automatic Garbage Collection စနစ်ပါဝင်တာကြောင့် memory leak ဖြစ်နိုင်ခြေနည်းပါတယ်။ Developer တွေအနေနဲ့ memory ကို manual manage လုပ်စရာမလိုပဲ အလိုအလျောက် ရှင်းလင်းပေးတာ ဖြစ်ပါတယ်။

အင်မတန်လုံခြုံစိတ်ချေရတဲ့ Programming language တစ်ခု ဖြစ်ပါတယ်။ Pointer concept မပါဝင်တာကြောင့် လုံခြုံမှုပိုမြင့်မားတယ်။ Security Manager နဲ့ Bytecode Verifier တို့ကြောင့် malicious code တွေကို ကောင်းကောင်း ကာကွယ်နိုင်တယ်။

Multithreading Support လုပ်ပေးတဲ့ အတွက် Java နဲ့ ရေးထားတဲ့ app တွေရဲ့ Performance တွေက အရမ်းကို smooth ဖြစ်ပါတယ်။ ပြောချင်တာက အရမ်းကို stable ဖြစ်ပြီး error တွေ application crash ဖြစ်တာတွေ အခြား language တွေထက်စာရင် အရမ်းနည်းပါတယ်။ Built-in multithreading support ပါဝင်တာကြောင့် concurrent programming လုပ်ရတာ အဆင်ပြေတယ်။ High-performance applications တွေဖန်တီးဖို့ အထောက်အကူဖြစ်စေတယ်။

Rich API Library ရှိတဲ့ language တစ်ခုဖြစ်ပါတယ်။ Networking, Database Connection, XML Processing, Utilities စတဲ့ built-in libraries အများအပြားရှိနေပါတယ်။ Third-party libraries တွေလည်း အများကြီးရှိနေတာကြောင့် ဘယ်လိုမျိုး app တွေရေးမလဲ၊ ဘယ်လိုမျိုး services တွေပေးမလဲ။ ဘာလာလာ အကုန်ဒေါင်းလို့ရတဲ့ စွယ်စုံသုံး programming language တစ်ခုဖြစ်ပါတယ်။



Enterprise-Level Application တွေအတွက်လည်း အထူးသင့်တော်ပါတယ်။ Banking systems, E-commerce platforms စိတ် large-scale applications တွေမှာ အများဆုံးအသုံးပြုကြတာ ဖြစ်ပါတယ်။ Spring, Hibernate စိတ် powerful frameworks တွေရှိပါတယ်။ Android Development အတွက်အခြေခံ အဖြစ်လွှဲလာထားသင့်တဲ့ language လည်းဖြစ်ပါတယ်။ Android OS က Java/Kotlin ကိုအခြေခံထားတာကြောင့် mobile app development အတွက်အရေးကြီးပါတယ်။

ကမ္ဘာအကြီးဆုံး Community Support ရှိပါတယ်။ StackOverflow, GitHub စိတ်နေရာတွေမှ help resources အများကြီးရှိပါတယ်။ Learning Career အတွက်ပဲ ဖြစ်ဖြစ် + Developer Career အတွက်ပဲ ဖြစ်ဖြစ် ပြဿနာတစ်ခုကြံးတိုင်း solution ရှာဖိုးအရမ်းလွယ်ကူပါတယ်။

Backward Compatibility ရှိတဲ့အတွက် နှစ်ပေါင်းများစွာ update လုပ်လာပေမယ့် ဆက်လက်ပြီး လာမယ့်အနာဂတ်မှာ java code တွေရဲ့ compatibility ကိုထိန်းသိမ်းထားပေးပါတယ်။ ဆိုလိုတာက ရှေးဟောင်း Java code တွေကိုပါ အခုထိ run နိုင်ဆဲဖြစ်ပါတယ်။ အဲဒါကြောင့် Java ဟာ နှစ်ပေါင်း 20 ကျော်ကြာ အသုံးပြုလာကြတဲ့အတွက် ရင့်ကျက်ပြီး stable ဖြစ်တဲ့ language တစ်ခုဖြစ်ပါတယ်။ ဒါကြောင့် enterprise applications တွေ၊ financial systems တွေ၊ government systems တွေမှာ Java ကို အဓိကထားအသုံးပြုကြတာဖြစ်ပါတယ်။



ECLIPSE INSTALLATION



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



ECLIPSE INSTALLATION

Eclipse IDE က Java Developer တွေအတွက် အထူးရည်ရွယ်ပြီး ထုတ်လုပ်ထားတဲ့အတွက် အကျိုးကျေးဇူးတွေ အများကြီး ရှိပါတယ်။ ပထမအချက်ကတော့ အခမဲ့နဲ့ Open Source ဖြစ်နေတာပဲဖြစ်ပါတယ်။ Eclipse က လုံးဝအခမဲ့ဖြစ်ပြီး Open Source ဖြစ်တာကြောင့် ငွေကြေးအကုန်အကျမရှိဘဲ သုံးနိုင်ပါတယ်။

Cross-Platform support လုပ်တဲ့အတွက် Windows, macOS, Linux စတဲ့ OS အမျိုးမျိုးမှာ အလုပ်လုပ်နိုင်ပါတယ်။ ပြီးပြည့်စုံတဲ့ Java Development Tools လည်းဖြစ်ပါတယ်။ Java SE, Java EE, Jakarta EE, Android Development စတာတွေအတွက် ပြည့်စုံတဲ့ Toolset ပါဝင်ပါတယ်။ Code Completion, Refactoring, Debugging စတဲ့ Features တွေ Built-in ပါပါတယ်။

Plugin Ecosystem ကြီးမားခြင်းကြောင့် Maven, Gradle, Git, Spring, Hibernate စတဲ့ Framework တွေနဲ့ အလွယ်တကူချိတ်ဆက်နိုင်ပါတယ်။ ကိုယ်လိုချင်တဲ့ Functionality တွေအတွက် Plugin တွေ အများကြီးရှိပါတယ်။ စွမ်းဆောင်ရည်မြင့်မားခြင်းကြောင့် ကြီးမားတဲ့ Project တွေကိုတောင် ထိတိရောက်ရောက် Manage လုပ်နိုင်ပါတယ်။

ဟုတ်ကဲ့ IntelliJ IDEA (အထူးသဖြင့် Ultimate Edition) က ပိုမိုကောင်းမွန်ပေမယ့် Paid Version ဖြစ်ပါတယ်။ NetBeans ကလည်း အခမဲ့ဖြစ်ပေမယ့် Eclipse လောက် Plugin Ecosystem မကြီးမားပါဘူး။ အဲဒါကြောင့် Java Beginner မှ Advanced Developer အထိ အထူးသဖြင့် Enterprise Java Development လုပ်သူများ၊ Plugin-based Custom Development လုပ်ချင်သူများ။ ဒီစာအုပ်မှာ Eclipse IDE ကို သုံးပြီး လေ့ကျင့်ခန်းတွေကို ရေးပြသွားမှာ ဖြစ်ပါတယ်။

Installation step တွေကို အောက်ပါအတိုင်း အဆင့်ဆင့် လုပ်ဆောင်ပါ။

Step 1:

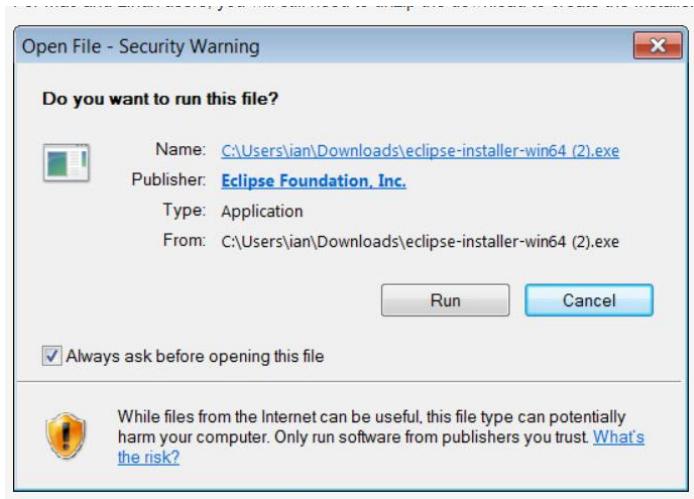
Eclipse installer ကို ဒေါင်းလုပ်ဆဲပါ။

Download Eclipse Installer from <http://www.eclipse.org/downloads>

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



Step 2:

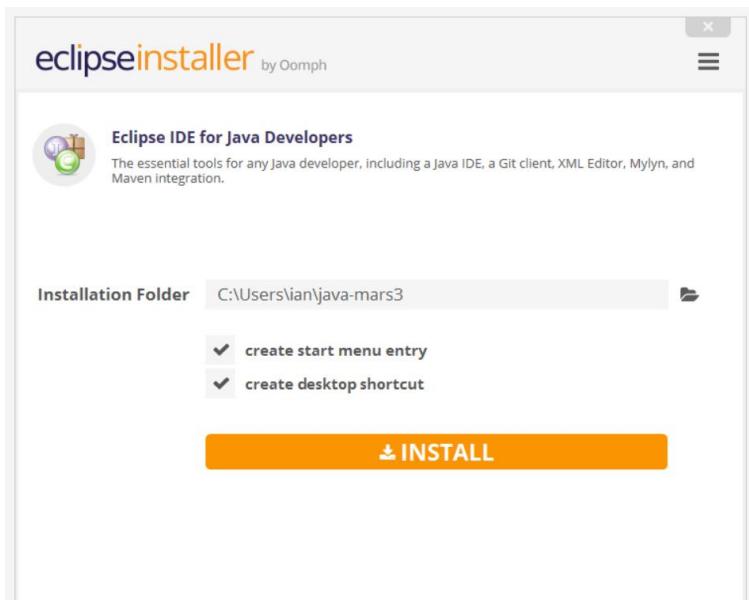


Step 3:



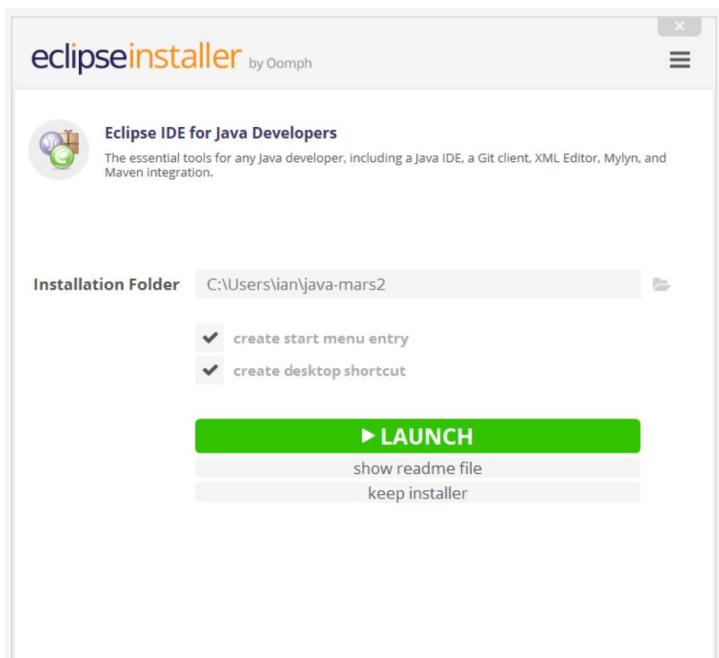


Step 4:



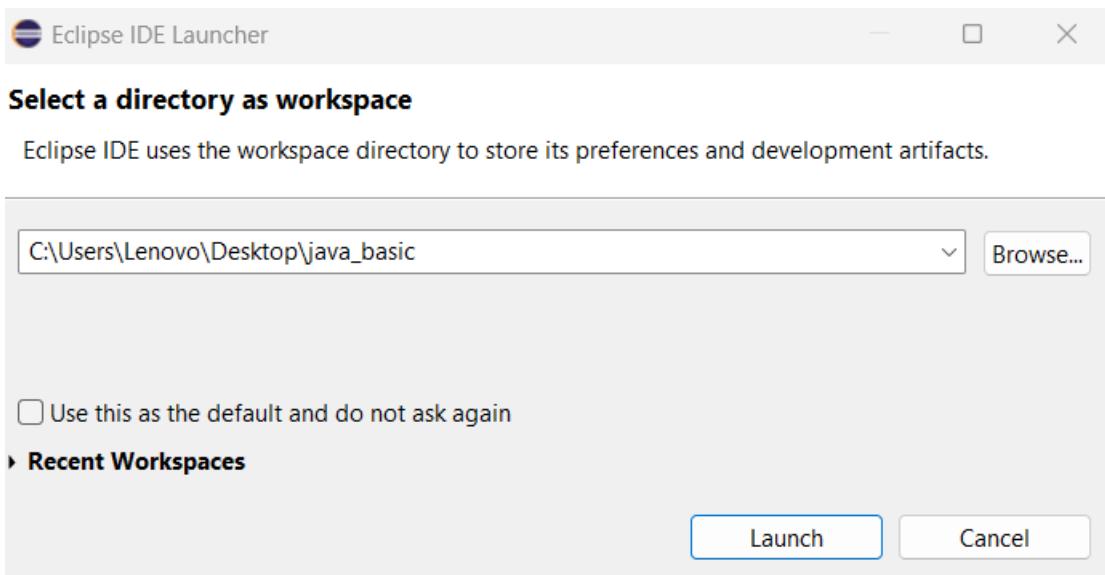
Step 5:

- በኋላም ማረጋገጫ ንግድ የሚያስፈልግ ይችላል



Step 6:

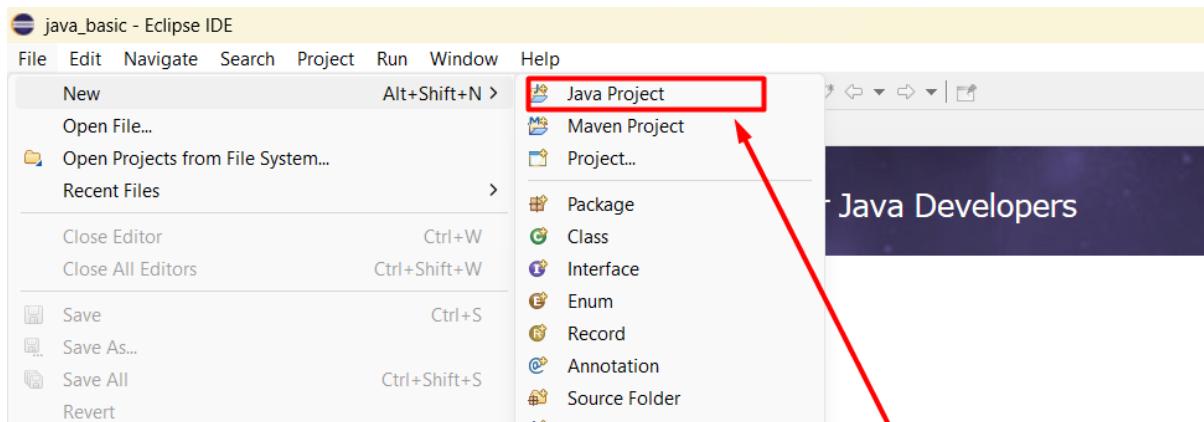
လေ့ကျင့်ခန်းတွေကို သိမ်းမယ့် location ကိုရွေးရပါမယ်။ အခု က desktop မှာ java_basic folder မှာ သိမ်းဖို့ ညွှန်းထားပါတယ်။



- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။

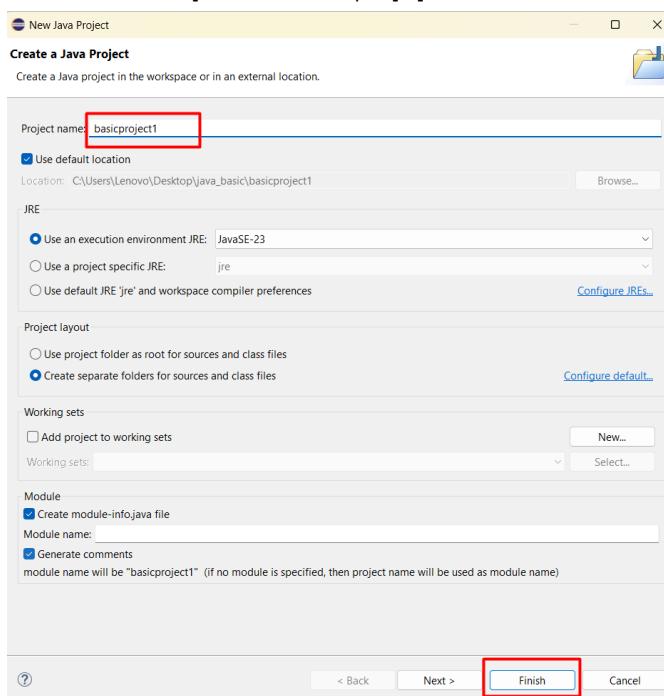


Step 7:



Step 8:

Project name የ small letter ተረጋግጣለን॥

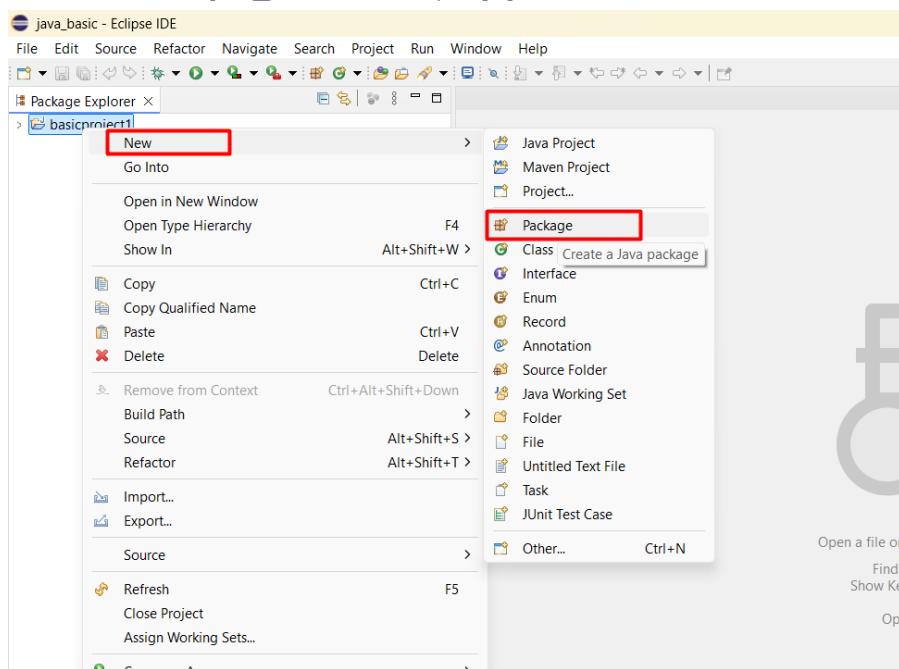


Step 9:

- በኩስ ማሸጊዎች የሚከተሉት ጥሩ ተመዝግበ ይችላል፡፡

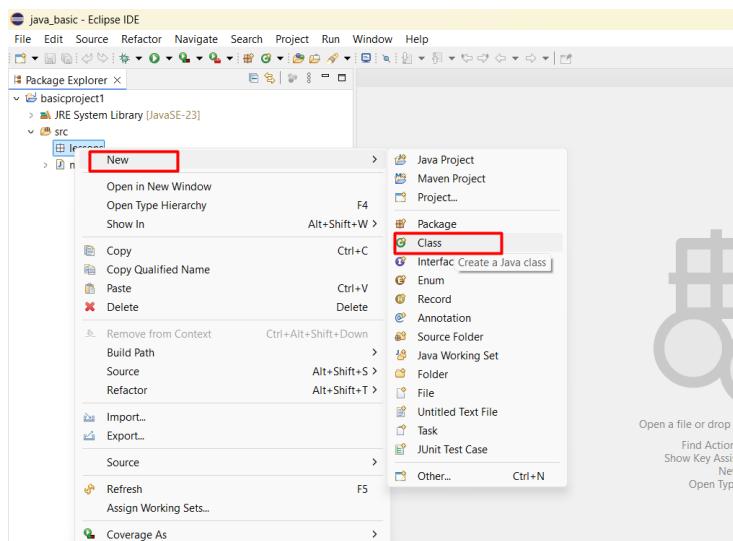


Package name ගිලවු: small letter න් රෙපිඡයි||

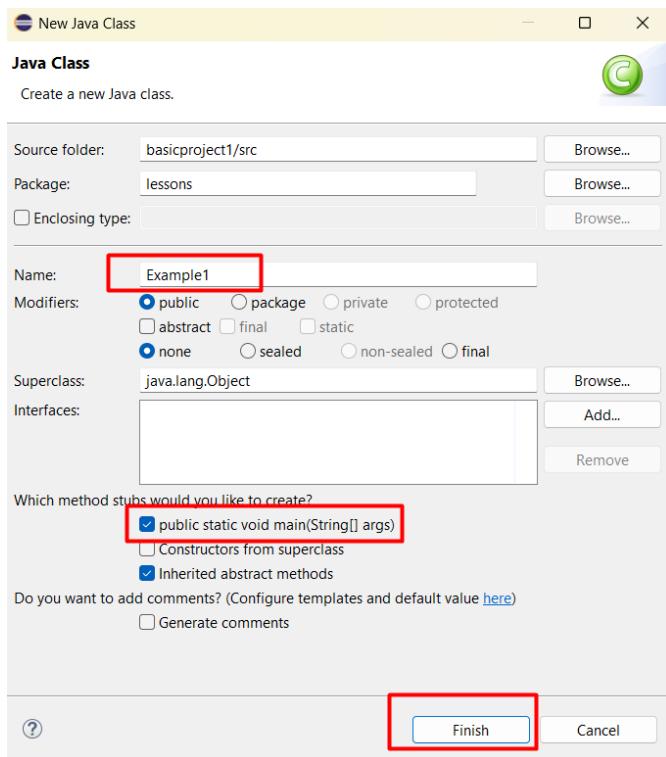


Step 10:

Class name ගිලෝතු capital letter න් රෙපිඡයි||



- මුද්‍රාප්‍රයෝගීක් සැක්සුනු තාක්ෂණික ප්‍රාග්ධනයි||



Step 11:

```
1 package lessons;
2
3 public class Example1 {
4
5     public static void main(String[] args) {
6         // TODO Auto-generated method stub
7
8             System.out.println("Hello Northern City...");
9
10    }
11
12 }
```



RUN

```
java_basic - basicproject1/src/lessons/Example1.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer X
basicproject1
  JRE System Library [JavaSE-23]
  src
    lessons
      Example1.java
      module-info.java
Example1.java X
1 package lessons;
2
3 public class Example1 {
4
5   public static void main(String[] args) {
6     // TODO Auto-generated method stub
7
8     System.out.println("Hello Northern City...");
9
10  }
11
12 }
13
```

Console X

```
terminated> Example1 [Java Application] C:\Users\Lenovo\p2\pool\plugins\org.eclipse.jst.j.openjdk.hots
Hello Northern City...
```

☞ Short Notes:

1. Project name ලේඛන සඳහා ප්‍රතිඵලියා යුතු කළ මෙහෙයුම් වේ||
2. Package name ලේඛන සඳහා ප්‍රතිඵලියා යුතු කළ මෙහෙයුම් වේ||
3. Class name ලේඛන සඳහා ප්‍රතිඵලියා යුතු කළ මෙහෙයුම් වේ||



DATA TYPES



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



Java မှာ Data Types ဆိုတာ variable တစ်ခုမှာ သိမ်းဆည်းမယ့် data အမျိုးအစားကို သတ်မှတ်ပေးတဲ့ အရာ ဖြစ်ပါတယ်။ Java မှာ Data Types ကို အဓိကအားဖြင့် မျိုးခွဲထားပါတယ်။

1. Primitive Data Types

ဒဲ Data Types တွေက အရမ်းရှိုးရှင်းပြီး မူလအတိုင်းပါဝင်တဲ့ အမျိုးအစားတွေဖြစ်ပါတယ်။

Data Type	Size	Default Value	Range	Example
byte	1 byte	0	-128 to 127	byte b = 100;
short	2 bytes	0	-32,768 to 32,767	short s = 5000;
int	4 bytes	0	-2 ³¹ to 2 ³¹ -1	int i = 100000;
long	8 bytes	0L	-2 ⁶³ to 2 ⁶³ -1	long l = 150000000000L;
float	4 bytes	0.0f	~6-7 decimal digits	float f = 5.75f;
double	8 bytes	0.0d	~15 decimal digits	double d = 19.99d;
char	2 bytes	'\u0000'	0 to 65,535	char c = 'A';
boolean	1 bit	false	true/false	boolean isJavaFun = true;

⇒ ဥပမာ Primitive Types အသုံးပြုပဲ့:

Code:

```
int age = 25;  
double salary = 500000.50;  
char grade = 'A';  
boolean isActive = true;
```

2. Non-Primitive Data Types

ဒဲ Data Types တွေက Object တွေကို ရည်ရွှေးပြီး Class တွေကနေ ဖန်တီးထားတာဖြစ်ပါတယ်။

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



Data Type	Description	Example
String	Character sequence	String name = "John";
Array	Fixed-size data structure	int[] numbers = {1, 2, 3};
Class	User-defined blueprint	Student stu = new Student();
Interface	Abstract type	List<String> list = new ArrayList<>();

⇒ ဥပမာ Non-Primitive Types အသုံးပြုပုံ:

Code:

```
String name = "Myanmar";
int[] marks = {80, 90, 85};
ArrayList<String> fruits = new ArrayList<>();
```

Primitive vs Non-Primitive Data Types

Feature	Primitive	Non-Primitive
အမျိုးအစား	Predefined	User-defined
အရွယ်အစား	Fixed	Dynamic
တန်ဖိုး	Stored directly	Store reference
Default Value	Yes	null
Usage	Faster	More features



☞ Short Notes:

1. Type Casting - int ကို double အဖြစ်ပြောင်းတာမျိုး

Code:

```
int num = 10;  
double dNum = num; // Automatic casting
```

2. Wrapper Classes - Primitive တွက် Object အဖြစ်သံဃွဲ

Code:

```
Integer num = 10; // Autoboxing  
int n = num; // Unboxing
```

3. String **မှာ** Special Case - Non-primitive ဖြစ်ပေမယ့် Java မှာ အထူးအဆင်ပြေအောင်လုပ်ထားတယ်

Java Data Types တွကို ကောင်းကောင်းနားလည်ထားရင် program တွေရေးတဲ့အခါ memory ကိုထိထိရောက်ရောက်သံဃွဲနိုင်မယ်၊ bug တွေလည်းနည်းမယ်။



VARIABLES



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



VARIABLES

Java မှာ variable ဆိုတာ data တွေထိမ်းဆည်းဖို့ memory location တစ်ခုကို နာမည်ပေးထားတာ ဖြစ်ပါတယ်။ Variable တိုင်းမှာ data type, name နဲ့ value တွေပါဝင်ပါတယ်။

□ Variable ကြော်နည်း ၃ မျိုး

1. Local Variables - Method တွေအတွင်းမှာသာ အသံးပြုနိုင်တယ်

Code:

```
public void calculate() {  
    int x = 10; // Local variable  
    System.out.println(x);  
}
```

2. Instance Variables - Class အတွင်းမှာကြော်ပြီး object တိုင်းအတွက် သီးသန်ရှိတယ်

Code:

```
class Student {  
    String name; // Instance variable  
    int age; // Instance variable  
}
```

3. Static Variables - Class level မှာရှိပြီး object အားလုံးအတွက် တူညီတယ်

Code:

```
class School {  
    static String schoolName = "ABC School"; // Static variable  
}
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



❑ Variable Case Styles

Java မှာ variable name တွေရေးတဲ့အခါ အောက်ပါ style တွေကို အသုံးပြုပါတယ်။

Case Style	ဥပမာ	အသုံးပြုပါ
camelCase	studentName, accountBalance	Variable/method names
PascalCase	Student, BankAccount	Class/Interface names
snake_case	MAX_SIZE, MIN_VALUE	Constants (final variables)
kebab-case	အသုံးမပြုပါ	Java မှာမသုံးပါ

✓ Variable Naming Rules

1. စာလုံးအသေး/အကြီး၊ \$ နဲ့ - နဲ့စနိုင်တယ် (ဂဏန်းနဲ့မစရ)

Code:

```
String name;
int _count;
double $price;
```

2. Java keywords တွေကို variable name အဖြစ်မသုံးရ

Code:

```
int class; // Error - 'class' is a keyword
```

3. Case sensitive ဖြစ်တယ်

Code:

```
String firstName;
String FirstName; // ဒါကဲ့ပြားတဲ့ variable
```

4. Meaningful names သုံးပါ

Code:

```
int a;      // မကောင်း
int age;    // ကောင်း
int userAge; // ပိုကောင်း
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



✓ Good vs Bad Variable Names

Code:

```
// Bad Examples
```

```
int a = 10;
```

```
String n = "John";
```

```
// Good Examples
```

```
int studentCount = 10;
```

```
String userName = "JohnDoe";
```

```
final double PI_VALUE = 3.14159; // Constant
```

□ သိတေသနများအရေးကြီးအချက်များ

- ✓ Variable name တွေကို camelCase နဲ့ရေးပါ
- ✓ Constants တွေကို UPPER_SNAKE_CASE နဲ့ရေးပါ
- ✓ Class name တွေကို PascalCase နဲ့ရေးပါ
- ✓ Meaningful names တွေထိုးပြီး abbreviation တွေရှေ့ပါ

⇒ ဥပမာ - ကောင်းမွန်တဲ့ variable names တွေ:

Code:

```
String firstName = "Kyaw Kyaw";
```

```
int maxScore = 100;
```

```
boolean isActive = true;
```

```
final double TAX_RATE = 0.05;
```

Variable naming convention တွေကို မှန်မှန်ကန်ကန်သုံးမယ်ဆိုရင် code တွေကို ဖတ်ရတာ ပိုလွယ်ကူမယ်၊
maintain လုပ်ရတာ ပိုအဆင်ပြေမယ်။



BASIC CALCULATIONS



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



BASIC CALCULATIONS

Java မှာ အခြေခံသချက်တွေ (ပေါင်း၊ နှုတ်၊ မြောက်၊ စား) လုပ်ဖို့နူးနာ program တစ်ခုရေးပြပါမယ်။ user ကနေ keyboard ကနေ number ၂ခုရှိက်ထည့်ပြီး ရလဒ်တွေကို ပြသပေးမယ်။

Addition:

Code:

```
import java.util.Scanner; // Scanner class ကို import လုပ်တယ်
```

```
public class Example1 {  
    public static void main(String[] args) {  
        // Scanner object ဖန်တီးတယ်  
        Scanner input = new Scanner(System.in);  
  
        // User ကနေ number ၂ခုရှိက်ထည့်ခိုင်းတယ်  
        System.out.print("Enter Num1: ");  
        int num1 = input.nextInt();  
  
        System.out.print("Enter Num2: ");  
        int num2 = input.nextInt();  
  
        // Calculation တွေလုပ်တယ်  
        int sum = num1 + num2;           // ပေါင်းခြင်း  
  
        // ရလဒ်တွေကိုပြသမယ်  
        System.out.println("\nThe Answer is :");  
        System.out.println(num1 + " + " + num2 + " = " + sum);  
  
        input.close(); // Scanner ကိုပိတ်တယ်  
    }  
}
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



Divide:

Code:

import java.util.Scanner; // Scanner class ကို import လုပ်တယ်

```
public class Example1 {  
    public static void main(String[] args) {  
        // Scanner object ဖန်တီးတယ်  
        Scanner input = new Scanner(System.in);  
  
        // User ကနေ number ချရှိက်ထည့်ခိုင်းတယ်  
        System.out.print("Enter Num1: ");  
        double num1 = input.nextDouble();  
  
        System.out.print("Enter Num2: ");  
        double num2 = input.nextDouble();  
  
        // Calculation တွေလုပ်တယ်  
        double quotient = num1 / num2;      // ပေါင်းခြင်း  
  
        // ရလဒ်တွေကိုပြသမယ်  
        System.out.println("\nThe answer is :");  
        System.out.println(num1 + " / " + num2 + " = " + quotient);  
  
        input.close(); // Scanner ကိုပိတ်တယ်  
    }  
}
```

▣ အသေးစိတ်ရှင်းလင်းချက်

1. Scanner Import လုပ်တယ်

Code:

```
import java.util.Scanner;  
o Keyboard input ယူနိုင် Scanner class လိုတယ်
```

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူဗာ တတ်ရမည်။



2. Scanner Object ဖုန်တီးတယ်

Code:

```
Scanner input = new Scanner(System.in);
```

3. User Input ယူတယ်

Code:

```
System.out.print( "Enter Num1: ");
```

```
double num1 = input.nextDouble();
```

- o nextDouble() method ဆုတေသနများတွင်ဖြစ်ပါသည်

4. တွက်ချက်မှုတွေလုပ်တယ်

Code:

```
double sum = num1 + num2;
```

5. ရလဒ်တွေပြုသမယ်

Code:

```
System.out.println(num1 + " + " + num2 + " = " + sum);
```

6. Scanner ဂိတ်တယ်

Code:

```
input.close();
```

⇒ ဥပမာ Run ကြည့်ရင်

Result:

Enter Num1: 15

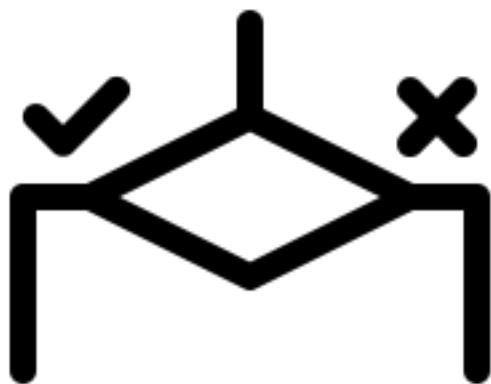
Enter Num2: 3

The answer is :

15.0 + 3.0 = 18.0



CONDITIONAL STATEMENTS



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



CONDITIONAL STATEMENTS

Java မှာ Conditional Statements (အခြေအနေစစ်ဆေးချက်များ) ဆိုတာ program ရဲလုပ်ဆောင်ချက်တွေကို အခြေအနေအလိုက် ပြောင်းလဲဖို့အတွက် အသုံးပြုပါတယ်။

Conditional Statements အမျိုးအစားများ

1. if Statement

Syntax:

```
if (condition) {  
    // condition မှန်ရင် ဒီcodeတွေအလုပ်လုပ်မယ်  
}
```

ဥပမာ:

Code:

```
int age = 18;  
if (age >= 18) {  
    System.out.println("You are eligible to vote now...");  
}
```

2. if-else Statement

Syntax:

```
if (condition) {  
    // condition မှန်ရင် ဒီcode  
} else {  
    // condition မှားရင် ဒီcode  
}
```

ဥပမာ:

Code:

```
int mark = 40;
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



```
if (mark >= 50) {  
    System.out.println("Pass...");  
} else {  
    System.out.println("Fail...");  
}
```

3.if-else if Ladder

Syntax:

```
if (condition1) {  
    // code 1  
} else if (condition2) {  
    // code 2  
} else {  
    // default code  
}
```

ឧបោះ:

Code:

```
int score = 75;  
if (score >= 80) {  
    System.out.println("Grade A");  
} else if (score >= 70) {  
    System.out.println("Grade B");  
} else if (score >= 60) {  
    System.out.println("Grade C");  
} else {  
    System.out.println("Grade D");  
}
```

4. Nested if

Syntax:

```
if (condition1) {  
    if (condition2) {  
        // code  
    }  
}
```

- មិនអាចលើកដែលគ្មាន់ពីក្រោមគ្មាន់ទាំងនេះទេ



ឧបោះ:

Code:

```
int age = 20;  
boolean hasID = true;
```

```
if (age >= 18) {  
    if (hasID) {  
        System.out.println("OK... Pass...");  
    } else {  
        System.out.println("Show Your ID Card");  
    }  
} else {  
    System.out.println("Sorry...You cannot enter...");  
}
```

5. switch Statement

Syntax:

```
switch (expression) {  
    case value1:  
        // code  
        break;  
    case value2:  
        // code  
        break;  
    default:  
        // default code  
}
```

ឧបោះ:

Code:

```
int day = 3;  
switch (day) {  
    case 1:  
        System.out.println("Monday");  
        break;  
    case 2:  
        System.out.println("Tuesday");  
}
```

- មិនអាចលើកឡាតាំងទេ ត្រូវប្រើពាក្យលេខ 1 ដែលមិនត្រូវបានសម្រេច



```
break;  
case 3:  
    System.out.println("Wednesday");  
    break;  
default:  
    System.out.println("Unknown Day...");  
}
```

Conditional Operator (Ternary Operator)

Syntax:

```
variable = (condition) ? expressionTrue : expressionFalse;
```

ဥပမာ:

Code:

```
int time = 20;  
String result = (time < 18) ? "Day" : "Night";  
System.out.println(result); // "Night"
```

☞ Short Notes:

1. Comparison Operators တွေသုံးပါ: ==, !=, >, <, >=, <=
2. Logical Operators တွေသုံးပါ: && (and), || (or), ! (not)
3. switch နဲ့ break မသုံးရင် fall-through ဖြစ်တတ်တယ်
4. Ternary Operator ကို simple conditions အတွက်သုံးပါ



လေ့ကျင့်ရန်

Code:

```
import java.util.Scanner;
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



```
public class WeatherAdvice {  
    public static void main(String[] args) {  
        Scanner input = new Scanner(System.in);  
        System.out.print("Enter Temperature: ");  
        int temp = input.nextInt();  
  
        if (temp > 30) {  
            System.out.println("The weather is so Hot.. Drink Water...");  
        } else if (temp > 20) {  
            System.out.println("The weather is fine...");  
        } else if (temp > 10) {  
            System.out.println("The weather is cold... Keep yourself warm...");  
        } else {  
            System.out.println("The weather is very cold...don't go outside...");  
        }  
  
        input.close();  
    }  
}
```

Conditional Statements တွက Java Programming မှာ အရေးကြီးတဲ့ အခြေခံဖြစ်ပြီး ဆုံးဖြတ်ချက်ချတဲ့ logic တွေရေးတဲ့အခါ အမြဲသုံးရပါမယ်။



လေ့ကျင့်ရန်

ဒီနှစ်နာမှာ user ကနေ ကဏ္ဍာ ၃ ခုရှိက်ထည့်ခိုင်းပြီး if-else statement သုံးကာ အကြီးဆုံးကဏ္ဍာကို ရှာပြပါမယ်။

Code:

```
import java.util.Scanner;
```

```
public class FindLargestNumber {  
    public static void main(String[] args) {
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



```
Scanner input = new Scanner(System.in);

// ଗଣକ: ୧ ଥାରିଗନ୍ତେ ଲେଖିବାରୁ ପାଇଁ
System.out.print("Enter Num1: ");
double a = input.nextDouble();

System.out.print("Enter Num2: ");
double b = input.nextDouble();

System.out.print("Enter Num3: ");
double c = input.nextDouble();

// ଅଟେଙ୍ଗିରେ ଗଣକରେ ଲେଖିବାରୁ ପାଇଁ
double largest;

if (a > b) {
    largest = a;
}
else {
    largest = b;
}

if(c>largest){
    Largest=c;
}
// ରହିବାରେ ପ୍ରାପ୍ତ ମହିନେ
System.out.println("\nThe largest no is : " + largest);

input.close();
}
```

▣ ଆପେକ୍ଷିତ କାମକାରୀ

1. Scanner ବ୍ୟାପିକୁ input ଦ୍ୱାରା ପାଇଁ

Code:

```
Scanner input = new Scanner(System.in);
```

- ଫର୍ମାଡ଼ୁ ପ୍ରକାଶିତ କରିବାକୁ ପାଇଁ



double a = input.nextDouble();
2. if-else condition තොක් ඇග්‍රිඩ්ස්ප්‍රින්මයි

Code:

```
if (a >= b && a >= c) {  
    largest = a;  
}  
else if (b >= a && b >= c) {  
    largest = b;  
}  
else {  
    largest = c;  
}
```

3. රුවරුවුමයි

Code:

```
System.out.println("\nThe largest no is : " + largest);
```

▪ උපහා Run හෝතුරු

Result:

Enter Num1: 25

Enter Num2: 50

Enter Num3: 10

The largest no is : 50.0

■ Advanced Version

Code:

```
import java.util.InputMismatchException;  
import java.util.Scanner;
```

```
public class SafeLargestNumberFinder {
```

■ මුද්‍රණයාටියින්: ගුණුවාට තරුණයුතු



```
public static void main(String[] args) {  
    Scanner input = new Scanner(System.in);  
  
    try {  
        System.out.print( "Enter Num1: ");  
        double a = input.nextDouble();  
  
        System.out.print( "Enter Num2: ");  
        double b = input.nextDouble();  
  
        System.out.print( "Enter Num3: ");  
        double c = input.nextDouble();  
  
        double largest;  
  
        if (a == b && b == c) {  
            System.out.println( "\nThe inputs no are the same...");  
            largest = a;  
        }  
        else if (a >= b && a >= c) {  
            largest = a;  
        }  
        else if (b >= a && b >= c) {  
            largest = b;  
        }  
        else {  
            largest = c;  
        }  
  
        System.out.println( "\n The largest no is : " + largest);  
  
    } catch (InputMismatchException e) {  
        System.out.println( "Invalid Inputs...");  
    } finally {  
        input.close();  
    }  
}
```

၃ Advanced Version မှာ အသစ်ထပ်ထည့်ထားတော်

■ မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



- ✓ ဂကန်းအားလုံးတူဖောင် special message ပြတ်
- ✓ Input မှားရင် error handling လုပ်ထားတာ (try-catch)
- ✓ Scanner ကို finally block မှာ ပိတ်ထားတာ

Conditional statements တွေကို ဒီလိုအခြေအနေတွေမှာ အသုံးဝင်ပါတယ်:

- User input validation
- Business logic decisions
- Game development (e.g., scoring systems)
- Algorithm implementations



လေ့ကျင့်ရန် -

Find the largest No

Enter Num1: 2

Enter Num2: 44

Enter Num3: 32

Enter Num4: 25

The largest no is : 44



LOOPS



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



LOOPS

Java မှာ Loops (repeat control) ဆိုတာ code တစ်ပိုင်းကို အထပ်ထပ် အခါခါ အလုပ်လုပ် စေနိုင် အသံးဖြုပါတယ်။ Loops တွေကို အမိကအားဖြင့် ၃ မျိုးခွဲထားပါတယ်။

1. for Loop

Syntax:

```
for (start; stop; increment/decrement) {  
    // code block statement  
}
```

ဥပမာ:

Code:

```
for (int i = 1; i <= 5; i++) {  
    System.out.println( "Hello " + i);  
}
```

ရလဒ်:

Output:

```
Hello 1  
Hello 2  
Hello 3  
Hello 4  
Hello 5
```

2. while Loop

Syntax:

```
while (condition) {  
    // condition မှန်နေသမှု ဆက်လုပ်မယ်  
}
```

ဥပမာ:

Code:

```
int count = 1;
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
while (count <= 3) {  
    System.out.println("Count: " + count);  
    count++;  
}
```

ရလဒ်:

Output:

Count: 1

Count: 2

Count: 3

3. do-while Loop

Syntax:

```
do {  
    // အနည်းဆုံး ၁ ခါတော့ အလုပ်လုပ်မယ်  
} while (condition);
```

ဥပမာ:

Code:

```
int x = 5;  
do {  
    System.out.println("Value: " + x);  
    x--;  
} while (x > 0);
```

ရလဒ်:

Copy

Value: 5

Value: 4

Value: 3

Value: 2

Value: 1



❑ Loops တွေရဲအထူးအချက်များ

break Statement

Code:

```
for (int i = 1; i <= 10; i++) {  
    if (i == 5) {  
        break; // loop ကနေ ထွက်သွားမယ်  
    }  
    System.out.println(i);  
}
```

continue Statement

Code:

```
for (int i = 1; i <= 5; i++) {  
    if (i == 3) {  
        continue; // ဒီအကြိမ်ကိုကျပ်မယ်  
    }  
    System.out.println(i);  
}
```

Nested Loops

Code:

```
for (int i = 1; i <= 3; i++) {  
    for (int j = 1; j <= 2; j++) {  
        System.out.println("i=" + i + ", j=" + j);  
    }  
}
```



□ ဘယ် Loop ကို ဘယ်အချိန်မှာ သုံးမလဲ?

Loop Type	အသုံးပြုရမယ့်အခြေအနေ
for	လုပ်ရမယ့်အကြိမ်ရေသိရင်
while	လုပ်ရမယ့်အကြိမ်ရေမသိရင်
do-while	အနည်းဆုံး ၁ ခါလုပ်ဖို့လိုရင်

Practice 1:

Code:

```
import java.util.Scanner;

public class MultiplicationTable {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a Number: ");
        int num = input.nextInt();

        System.out.println(num + " table list ...");
        for (int i = 1; i <= 10; i++) {
            System.out.println(num + " x " + i + " = " + (num*i));
        }

        input.close();
    }
}
```

☞ Short Notes:

- Loops တွေကို array/list တွေနဲ့ တွဲသုံးလေ့ရှိတယ်
- Infinite loop (အဆုံးမရှိတဲ့ loop) ဖြစ်မသွားအောင် ဂရစိုက်ပါ
- break နဲ့ continue တွေကို လိုအပ်မှသာသုံးပါ

Loops တွေက Java Programming မှာ အရေးပါတဲ့ အခြေခံ concept တစ်ခုဖြစ်ပြီး data တွေကို အထပ်ထပ်လုပ်ဆောင်ဖို့လိုတဲ့ အခါတိုင်း အသုံးပြုရပါမယ်။

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



ଲେଖିବାର୍ଷିକ-

Practice 1:

i=10

j=20

```
#####
#          #
#          #
#          #
#####
#  #      #  #
#  #      #  #
#  #      #  #
#  #      #  #
#####

```



Practice 2:

i=11

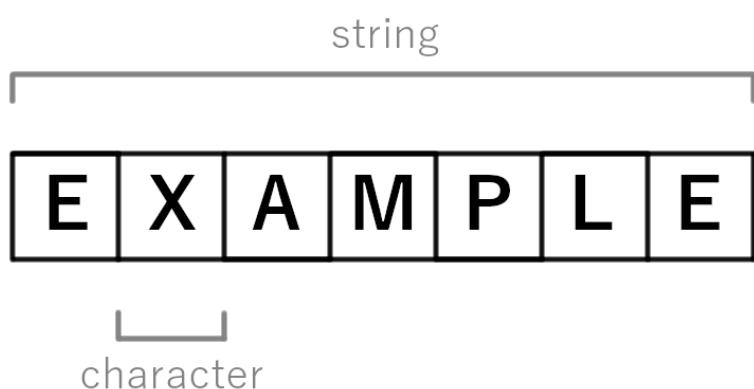
j=11

```
#####
#    #    #
#    ###    #
#    #####   #
#    ####### #
#####
#    ####### #
#    ####### #
#    ###    #
#    ##    #
#    #    #
#####

```



STRING MANIPULATIONS



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



STRING MANIPULATIONS

Java မှာ String တွက် လွယ်လွယ်ကူကူ အသုံးပြန်ည်းနဲ့ လုပ်ဆောင်ချက်တွက် ရှင်းပြပေးမယ်။

1. String ဖန်တီးနည်း

Code:

```
String str1 = "Hello";      // String Literal (Memory-efficient)  
String str2 = new String("Java"); // new Keyword (Heap Memory)
```

2. String Methods အသုံးပြန်ည်း

(a) Length & Case ပြောင်းခြင်း

Code:

```
String text = "Java Programming";  
System.out.println(text.length()); // 16 (စာလုံးရေ)  
System.out.println(text.toUpperCase()); // "JAVA PROGRAMMING"  
System.out.println(text.toLowerCase()); // "java programming"
```

(b) String ပေါင်းခြင်း (Concatenation)

Code:

```
String s1 = "Hello" ;  
String s2 = "Java" ;  
System.out.println(s1 + " " + s2); // "Hello Java"  
System.out.println(s1.concat(s2)); // "HelloJava"
```

(c) String နှင့်ယူနှိပ်ခြင်း (Comparison)

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



Code:

```
String a = "Java" ;
String b = "java" ;
System.out.println(a.equals(b)); // false (Case-sensitive)
System.out.println(a.equalsIgnoreCase(b)); // true
System.out.println(a.compareTo(b)); // -32 (ASCII diff)
```

(d) Substring & Character ဖြတ်ယူခြင်း

Code:

```
String str = "Programming";
System.out.println(str.substring(3)); // "gramming"
System.out.println(str.substring(3, 7)); // "gram"
System.out.println(str.charAt(4)); // 'r' (Index 4)
```

(e) String Search & Replace

Code:

```
String msg = "I love Java";
System.out.println(msg.contains("Java")); // true
System.out.println(msg.indexOf("love")); // 2
System.out.println(msg.replace("Java", "Python")); // "I love Python"
```

(f) String Split & Join

Code:

```
String data = "Apple,Banana,Mango" ;
String[] fruits = data.split( "," ); // [ "Apple" , "Banana" , "Mango" ]
String joined = String.join( " - ", fruits); // "Apple-Banana-Mango"
```

3. StringBuilder vs StringBuffer

- `StringBuilder` → မြန်ဆန် (Thread-safe မဟုတ်)
- `StringBuffer` → Thread-safe (နောက်လေမယ့် Multi-thread အဆင်ပြု)

Code:

```
StringBuilder sb = new StringBuilder("Hello");
sb.append(" Java");
System.out.println(sb.toString()); // "Hello Java"
```



4. Escape Characters

Code:

```
System.out.println("He said, \"Hello!\""); // "He said, "Hello!""
System.out.println("Line1\nLine2"); // New Line
System.out.println("Tab\tSpace"); // Tab
```

5. String Formatting

Code:

```
String name = "Alice";
int age = 25;
System.out.printf("Name: %s, Age: %d", name, age);
System.out.println(String.format("Name: %s", name));
```

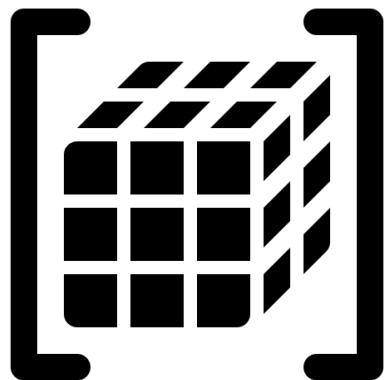
□ Short Notes

- Java String သည် Immutable (ပြောင်းလဲခြုံမရ) ဖြစ်တောက်ကြာင့် StringBuilder/StringBuffer ကို
မကြေခဲ့ပြုပြင်ရန် လိုအပ်ပါက သုံးသင့်ပါတယ်။

Java String Manipulation သည် Project တိုင်းအတွက် အရေးကြီးပြီး ဒီ Methods တွေကို ကျေမှုးကျင်စွာ
အသုံးပြုနိုင်ရန် Coding Efficiency တက်မှာပါ!



STATIC ARRAY



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



STATIC ARRAY

Static Array ဆိတ် Fixed Size ရှိတဲ့ Array တစ်မျိုးဖြစ်ပြီး Java မှာ အောက်ပါအတိုင်း အသံးပြုနိုင်ပါတယ်။

1. Static Array ကို ဘယ်လိုသတ်မှတ်မလဲ?

⇒ Declaration & Initialization

Code:

```
// နည်းလမ်း (၁) - Size သတ်မှတ်ပြီး နောက်မှတန်ဖိုးထည့်ခြင်း  
int[] numbers = new int[5];  
numbers[0] = 10;  
numbers[1] = 20;  
  
// နည်းလမ်း (၂) - တန်ဖိုးတွေကို တစ်ခါတည်းထည့်ခြင်း  
String[] names = {"Alice", "Bob", "Charlie"};
```

⇒ Array Length သိရှိခြင်း

Code:

```
System.out.println(names.length); // 3 (အရေအတွက်)
```

2. Static Array ရဲ့ အရေးကြီးသော Features

⇒ Fixed Size

- တစ်ခါသတ်မှတ်ပြီးရင် Size ပြောင်းလျှော့မရ
- `ArrayIndexOutOfBoundsException` ဖြစ်နိုင်တယ် (အကယ်၍ Invalid Index သုံးမိရင်)

⇒ Fast Access ($O(1)$ Time Complexity)

Index သုံးပြီး တန်ဖိုးတွေကို ချက်ချင်းရှာနိုင်တယ်

Code:

```
System.out.println(names[1]); // "Bob"
```

⇒ Primitive & Reference Types နှစ်မျိုးလုံးအတွက် အသံးပြုနိုင်တယ်

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



Code:

```
int[] ages = {20, 25, 30}; // Primitive (int)  
String[] cities = {"Yangon", "Mandalay"}; // Reference (String)
```

3. Static Array ကို Loop ပတ်ခြင်း

⇒ For Loop သုံးခြင်း

Code:

```
for (int i = 0; i < numbers.length; i++) {  
    System.out.println(numbers[i]);  
}
```

⇒ Enhanced For Loop (For-Each)

Code:

```
for (String name : names) {  
    System.out.println(name);  
}
```

4. Static Array vs Dynamic Array (ArrayList)

Feature	Static Array	ArrayList (Dynamic)
Size	Fixed (ပြောင်းလဲရှုမရ)	Resizable (အလိုအလျောက်ကြိုးထွားနိုင်)
Performance	ပိုမြန်တယ် (Memory-efficient)	နည်းနည်းနေးတယ် (ဒါပေမယ့် Flexible)
Methods	length, Index-based Access	add(), remove(), get(), size()

5. Static Array သုံးသင့်တဲ့ အချိန်

- အချိန်အစားသေချာသိပြီး မပြောင်းလဲတဲ့ အခါမျိုး (ဥပမာ - Days of the Week)
- အမြန်ဆုံး Access လိုအပ်တဲ့ အခါမျိုး

ဥပမာ - Days of the Week

Code

```
String[] days = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};  
System.out.println(days[3]); // "Thu"
```



□ Short Notes

- Java မှာ Array က Object ဖြစ်တာကြောင့် new keyword သုံးပြီး Memory မှာ နေရာယူရတယ်။
- Static Array ကိုအချက်အထားသေချာသိရင် သုံးပါ၊ မသိရင် ArrayList သုံးပါ။



လောက်ပွဲ

□ Java Static Number Array Application

အောက်ပါ program မှာ static number array တစ်ခုကို manage လုပ်ဖို့ Java application တစ်ခုဖြစ်ပါတယ်။
addNewNum, showNum, sortNum functions တွေပါဝင်ပါတယ်။

Code:

```
import java.util.Arrays;
import java.util.Scanner;

public class NumberArrayApp {
    // Static array to store numbers
    private static int[] numbers = new int[10];
    private static int count = 0;
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        boolean running = true;

        while (running) {
            System.out.println("\nNumber Array Application Menu:");
            System.out.println("1. Add New Number");
            System.out.println("2. Show All Numbers");
            System.out.println("3. Sort Numbers");
            System.out.println("4. Exit");
            System.out.print("Enter your choice: ");
            int choice = scanner.nextInt();

            switch (choice) {
                case 1:
                    addNewNum();
                    break;
                case 2:
                    showNum();
                    break;
                case 3:
                    sortNum();
                    break;
                case 4:
                    running = false;
                    break;
                default:
                    System.out.println("Invalid choice!");
            }
        }
    }

    private static void addNewNum() {
        System.out.print("Enter a number: ");
        int num = scanner.nextInt();
        numbers[count] = num;
        count++;
    }

    private static void showNum() {
        System.out.println("All numbers in the array are:");
        for (int i = 0; i < count; i++) {
            System.out.print(numbers[i] + " ");
        }
    }

    private static void sortNum() {
        Arrays.sort(numbers);
        System.out.println("The sorted array is:");
        for (int i = 0; i < count; i++) {
            System.out.print(numbers[i] + " ");
        }
    }
}
```



```
System.out.println("1. Add new number");
System.out.println("2. Show all numbers");
System.out.println("3. Sort numbers");
System.out.println("4. Exit");
System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

switch (choice) {
    case 1:
        addNewNum();
        break;
    case 2:
        showNum();
        break;
    case 3:
        sortNum();
        break;
    case 4:
        running = false;
        System.out.println("Program ම දැක්මයි... ");
        break;
    default:
        System.out.println("මායුදුවේ රුපුවූ! 1 ම 4 පිටත මායුදුවේ රුපුවි");
}
}

scanner.close();
}

// Function to add new number to the array
private static void addNewNum() {
    if (count >= numbers.length) {
        System.out.println("Array පූරුෂ පිළිවුරුව සඳහා නොමැතිවා!");
        return;
    }

    System.out.print("නැවත තුළු මායුදුවේ මායුදුවි: ");
    int newNumber = scanner.nextInt();
```



```
numbers[count] = newNumber;
count++;
System.out.println(newNumber + " ወደ array ተወስኝ ይሆናል፡በሚችለ");
}

// Function to show all numbers in the array
private static void showNum() {
    if (count == 0) {
        System.out.println("Array ተወስኝ የሚገኘውን አንድነት ይሆናል፡");
        return;
    }

    System.out.println("Array ተወስኝ የሚገኘውን አንድነት ይሆናል፡");
    for (int i = 0; i < count; i++) {
        System.out.print(numbers[i] + " ");
    }
    System.out.println();
}

// Function to sort the numbers in the array
private static void sortNum() {
    if (count == 0) {
        System.out.println("Array ተወስኝ የሚገኘውን አንድነት ይሆናል፡");
        return;
    }

    // Create a copy of the array with only the filled elements
    int[] tempArray = Arrays.copyOf(numbers, count);

    // Sort the temporary array
    Arrays.sort(tempArray);

    // Copy the sorted elements back to the original array
    System.arraycopy(tempArray, 0, numbers, 0, count);

    System.out.println("የሚከተሉት አንድነት ይሆናል፡");
}

}
```



▣ အသေးစိတ်ရှင်းလင်းချက်

1. Static Array:

- `private static int[] numbers = new int[10];` - နံပါတ်များကို သိမ်းဆည်းဖို့ static array တစ်ခုဖြစ်ပါတယ်။
- `private static int count = 0;` - array ထဲမှာ ရှိတဲ့ element အရေအတွက်ကို ခြေရာခံဖို့ဖြစ်ပါတယ်။

2. addNewNum() function:

- User ထဲမှ နံပါတ်တစ်ခုကို ရယူပြီး array ထဲသို့ ထည့်သွင်းပေးပါတယ်။
- Array ပြည့်နေပါက နောက်ထပ်ထည့်၍ မရကြာင်း အကြောင်းကြားပါတယ်။

3. showNum() function:

- Array ထဲရှိ နံပါတ်များကို ဖော်ပြပေးပါတယ်။
- Array ထဲတွင် ဘာမှမရှိသေးပါက အကြောင်းကြားစာ ပြသပါတယ်။

4. sortNum() function:

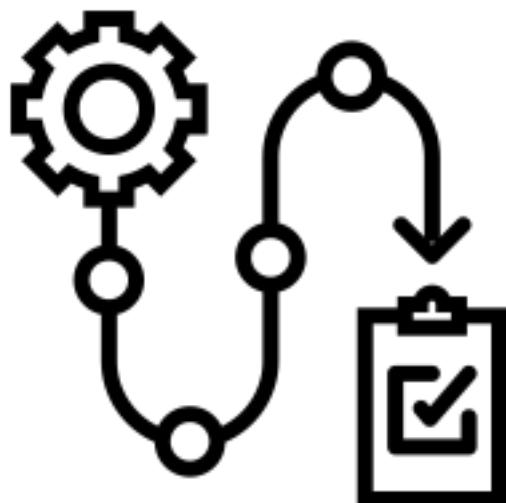
- Array ထဲရှိ နံပါတ်များကို စီပေးပါတယ်။
- Java ရဲ့ built-in Arrays.sort() method ကို အသုံးပြုထားပါတယ်။

5. Menu System:

- User ကို option 4 ခုပါတဲ့ menu တစ်ခု ပြသပါတယ်။
- User ရှေးချယ်မှုအလိုက် အထက်ပါ functions တွေကို ခေါ်ယူအသုံးပြုပါတယ်။



METHODS



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



Methods

Java မှာ Methods (မှတ်သားလုပ်ဆောင်ချက်များ) ဆိုတာ code တွေကို အပိုင်းလိုက်ခွဲပြီး ပြန်လည်အသုံးပြန်စေအောင် ဖန်တီးထားတဲ့ block တစ်ခုဖြစ်ပါတယ်။

□ Method ရဲ့ အခြေခံဖွဲ့စည်းပုံ

Syntax:

```
accessModifier returnType methodName(parameters) {  
    // method body  
    return value; // return type ရှိရင်  
}
```

□ Method အမျိုးအစားများ

1. Parameter မပါတဲ့ Method

Code:

```
public void greet() {  
    System.out.println("မင်္ဂလာပါ!");  
}
```

```
// ခေါ်ပြီးပုံ  
greet(); // Output: မင်္ဂလာပါ!
```

2. Parameter ပါတဲ့ Method

Code:

```
public void add(int a, int b) {  
    System.out.println("ပေါင်းလဒ်: " + (a+b));  
}
```

```
// ခေါ်ပြီးပုံ
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



```
add(5, 3); // Output: പീംഗ്: 8
```

3. Return Value പിന്തു സെമ്പ്പ് Method

Code:

```
public int square(int num) {  
    return num * num;  
}
```

```
// ഓട്ടിംഗ്  
int result = square(4);  
System.out.println(result); // Output: 16
```

4. Static Method

Code:

```
public class Calculator {  
    public static int multiply(int x, int y) {  
        return x * y;  
    }  
}
```

```
// ഓട്ടിംഗ് (object മാറ്റിയാണ്)  
int ans = Calculator.multiply(5, 6);  
System.out.println(ans); // Output: 30
```

❑ Method Overloading

ത്രാസ്റ്റു method name കുറഞ്ഞുള്ള parameter അല്ലാം മറ്റൊരു method നേരിട്ടിൽ വരുന്നതായാണ് മുൻകണ്ണിക്കാൻ.

Code:

```
public class Display {  
    public void show(int num) {  
        System.out.println("ഒന്ത്: " + num);  
    }  
  
    public void show(String text) {  
        System.out.println("ശബ്ദം: " + text);  
    }  
}
```

- ഫ്രെംബാപ്രവർത്തനാംഗം കൂട്ടുമാറ്റം തന്നെ മാറ്റം.



```
public void show(double num) {  
    System.out.println("အသာမဂတန်း: " + num);  
}  
}  
  
// ခေါ်သုံးပဲ  
Display d = new Display();  
d.show(10); // ဂဏန်း: 10  
d.show("Hello"); // စာသား: Hello  
d.show(3.14); // အသာမဂတန်း: 3.14
```

Recursive Method (ပြန်ခေါ်သုံးနိုင်သော Method)

Code:

```
public int factorial(int n) {  
    if (n == 1) return 1;  
    return n * factorial(n-1);  
}
```

// ခေါ်သုံးပဲ

```
System.out.println(factorial(5)); // Output: 120 (5!)
```

▣ Method သုံးရတဲ့ အကျိုးကျေးဇူးများ

1. Code Reusability - တစ်ခါရေးပြီး အကြိမ်ကြိမ်သုံးနိုင်တယ်
2. Modularity – Code တွေကို အပိုင်းလိုက်ခွဲနိုင်တယ်
3. Maintainability - ပြင်ဆင်ရတာ ပိုလွယ်တယ်
4. Abstraction – Complex logic တွေကို ဖုံးကွယ်နိုင်တယ်



Practice 1:

Code:

```
import java.util.Scanner;

public class MethodExample {
    // BMI တွက်တဲ့ method
    public static double calculateBMI(double weight, double height) {
        return weight / (height * height);
    }

    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter Weight (kg): ");
        double w = input.nextDouble();

        System.out.print("Enter Height (m): ");
        double h = input.nextDouble();

        double bmi = calculateBMI(w, h);
        System.out.printf("Your BMI: %.2f ", bmi);

        input.close();
    }
}
```

☞ Short Notes:

- void return type ဆိုရင် return statement မလိုဘူး
- Method name တွက် camelCase နဲ့ရေးပါ (ဥပမာ - calculateTotal)
- Parameter တွက် comma ခံပြီးသတ်မှတ်ပါ
- Java မှာ Pass by Value သာရှိတယ် (Pass by Reference မရှိဘူး)



Methods တွက Java Programming မှာ အရေးကြီးတဲ့ building blocks တွေဖြစ်ပြီး program တွကို ပိုမိုစနစ်ကျအောင် ရေးသားနိုင်ဖို့ ကူညီပေးပါတယ်။



လေ့ကျင့်ရန် -

Practice 1:

MENU

1.Add

2.Subtract

3. Exit



OOP



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။

 OOP

Java မှ OOP ဆိတ် Object-Oriented Programming ပါ။ ဒါက real-world entities တွေကို objects အဖြစ် model လုပ်ပြီး program တွေရေးသားတဲ့နည်းလမ်းတစ်ခုပါ။ OOP က program တွေကို ပိုပြီး organized, reusable နဲ့ maintainable ဖြစ်အောင် ကူညီပေးပါတယ်။ Java OOP က Encapsulation, Inheritance, Polymorphism, Abstraction ဆိတ် အခြေခံ concepts (4 pillars) တွေပေါ်မှာ အခြေခံထားပါတယ်။ Real-world problems တွေကို objects အဖြစ် model လုပ်ရာမှာ အများကြီး အထောက်အကြပ်ပါတယ်။ အသေးစိတ်ကို ဆက်လက်ပြီး လေ့လာကြည့်ရအောင်။

 Constructor

Java မှ Constructor ဆိတ်က Object တစ်ခုကို အသစ်ဖန်တီးတဲ့အခါ အလိုအလျောက်ခေါ်တဲ့ special method တစ်ခုပါ။ Constructor က object တွေရဲ့ ကနဦးအခြေအနေ (initial state) ကို သတ်မှတ်ဖို့အတွက် အသုံးပြုပါတယ်။

 □ Constructor ရဲ့ အဓိက အချက်များ

- Class နာမည်နဲ့ တူရမယ်
- Return type မထည့်ရဘူး (void, int စတာတွေ မပါဘူး)
- new keyword သုံးပြီး object ဖန်တီးတဲ့အခါ အလိုအလျောက်အလုပ်လုပ်တယ်
- Overloading လုပ်လို့ရတယ် (Parameter အမျိုးမျိုးနဲ့ Constructor အများကြီးသတ်မှတ်နိုင်တယ်)

 □ Constructor အမျိုးအစားများ

1. Default Constructor (Parameter မပါတဲ့ Constructor)

- Java Compiler က အလိုအလျောက် ဖန်တီးပေးတယ် (ဒါမှမဟုတ် ကိုယ်တိုင်လည်းရေးလို့ရတယ်)
- Instance variables တွေကို default values (int=0, String=null) တွေနဲ့ စပါတယ်

ဥပမာ:

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



Code:

```
class Phone {  
    String brand;  
  
    // Default Constructor  
    Phone() {  
        brand = "Unknown";  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Phone myPhone = new Phone(); // Constructor ကိုခေါ်တယ်  
        System.out.println(myPhone.brand); // Output: Unknown  
    }  
}
```

2. Parameterized Constructor (Parameter ပါတဲ့ Constructor)

- Object ဖန်တီးတဲ့အခါ တန်ဖိုးတွေပို့ပြီး အစပြုလုပ်နိုင်တယ်

ဥပမာ:

Code:

```
class Student {  
    String name;  
    int age;  
  
    // Parameterized Constructor  
    Student(String n, int a) {  
        name = n;  
        age = a;  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Student stu1 = new Student("Su Myat", 18);  
        System.out.println(stu1.name + " age is :" + stu1.age);  
        //Output: Su Myat age is: 18
```



```
}
```

3. Constructor Overloading (Constructor အများကြွေး)

- တူညီတဲ့ Class ထဲမှ Parameter အမျိုးမျိုး နဲ့ Constructor တွေအများကြွေးသတ်မှတ်နိုင်တယ်

ဥပမာ:

Code:

```
class Laptop {  
    String brand;  
    double price;  
  
    // Constructor 1: No-args  
    Laptop() {  
        brand = "Default";  
        price = 0.0;  
    }  
  
    // Constructor 2: Brand only  
    Laptop(String b) {  
        brand = b;  
        price = 0.0;  
    }  
  
    // Constructor 3: Full details  
    Laptop(String b, double p) {  
        brand = b;  
        price = p;  
    }  
  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Laptop lap1 = new Laptop(); // Constructor 1  
        Laptop lap2 = new Laptop("Lenovo"); // Constructor 2  
        Laptop lap3 = new Laptop("MacBook", 1500.0); // Constructor 3  
    }  
}
```

**□ this Keyword သုံးနည်း**

- Instance variable နဲ့ Parameter အမည်တူဖော်ရင် this သုံးပြီးခွဲ့ခြားနိုင်တယ်

ဥပမာ:

Code:

```
class Person {  
    String name;  
  
    Person(String name) {  
        this.name = name; // this.name = instance variable, name = parameter  
    }  
}
```

□ super(): Parent Class Constructor ခေါ်နည်း

- Inheritance မှာ Child Class Constructor က Parent Class Constructor ကို super() နဲ့ခေါ်နိုင်တယ်
- super() က Constructor ရဲ့ပထမဆုံး statement ဖြစ်ရမယ်

ဥပမာ:

Code:

```
class Vehicle {  
    Vehicle() {  
        System.out.println("Vehicle Constructor");  
    }  
}
```

```
class Car extends Vehicle {  
    Car() {  
        super(); // Vehicle Constructor ကိုခေါ်တယ်  
        System.out.println("Car Constructor");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {  
        Car myCar = new Car();  
        /* ရလဒ်:  
           Vehicle Constructor
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူဗာ တတ်ရမည်။



```
Car Constructor  
*/  
}  
}
```

▣ Short Notes:

- ✓ Constructor ကဲ Object ဖန်တီးတိုင်း အလိုအလျောက်အလုပ်လုပ်တယ်
- ✓ Constructor ကို ပုံမှန် method တွေလို ခေါ်လိုမရဘူး (new နဲ့ခေါ်ရတယ်)
- ✓ Constructor ကို private လုပ်ပြီး Singleton Pattern တွေမှာအသုံးပြုနိုင်တယ်

▣ Encapsulation

Encapsulation (Data Hiding) ဆိုတာ Java OOP ရဲ့အရေးကြီးတဲ့ concept တစ်ခုဖြစ်ပြီး data နဲ့ methods တွေကို တစ်စုတစ်စည်းတည်း ထုပ်ပိုးကာ အပြင်ကနေ တိုက်ရိုက်ဝင်ရောက်မှုကို ကန့်သတ်တဲ့နည်းလမ်း ဖြစ်ပါတယ်။

▣ Encapsulation ရဲ့အဓိက အချက်များ

1. Data hiding - Class အတွင်းရှိ data တွေကို private လုပ်ထားခြင်း
2. Controlled access - Getter/Setter methods တွေကနေ တဆင့်သာ data တွေကို access လုပ်ခြင်း
3. Increased security - Data တွေကို မလိုလားအပ်တဲ့ ပြောင်းလဲမှုတွေကနေ ကာကွယ်ပေးခြင်း

⇒ Encapsulation ကို ဘယ်လို Implement လုပ်မလဲ?

1. Variables တွေကို private လုပ်ပါ

Code:

```
public class Student {  
    private String name; // private variable  
    private int age; // private variable  
}
```

2. Getter/Setter Methods တွေ ဖန်တီးပါ

Code:

```
public class Student {
```

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
private String name;  
private int age;  
  
// Getter for name  
public String getName() {  
    return name;  
}  
  
// Setter for name  
public void setName(String name) {  
    this.name = name;  
}  
  
// Getter for age  
public int getAge() {  
    return age;  
}  
  
// Setter for age with validation  
public void setAge(int age) {  
    if(age > 0) { // Age validation  
        this.age = age;  
    } else {  
        System.out.println("အသက်က အနုတ်တန်ဖိုး မဖြစ်ရဘူး");  
    }  
}
```

3. Main Class မှာ အသုံးပြုပဲ

Code:

```
public class Main {  
    public static void main(String[] args) {  
        Student student = new Student();  
  
        // Set values using setters  
        student.setName( "မောင်မောင်");  
        student.setAge(20);
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
// Get values using getters  
System.out.println( "အမည်: " + student.getName());  
System.out.println( "အသက်: " + student.getAge());  
  
// Invalid age example  
student.setAge(-5); // Error message will be shown  
}  
}
```

❑ Encapsulation ရဲအကျိုးကျေးဇူးများ

1. Data Security - Data တွက် unauthorized access ကနေ ကာကွယ်ပေးတယ်
2. Flexibility - Internal implementation ကို ပြောင်းလဲရင် အပြင်က code တွက် မထိခိုက်စေဘူး
3. Validation - Data တွေထည့်တဲ့အခါ validation rules တွေ ထည့်ပေးလို့ရတယ်
4. Easy Maintenance - Code တွက် ပြန်ပြင်ရတာ ပိုလွယ်ကူတယ်

❑ တကယ့် Project တွေမှာ Encapsulation အသုံးပြုပုံ

Code:

```
public class BankAccount {  
    private String accountNumber;  
    private double balance;  
  
    public BankAccount(String accountNumber) {  
        this.accountNumber = accountNumber;  
        this.balance = 0.0;  
    }  
  
    public void deposit(double amount) {  
        if(amount > 0) {  
            balance += amount;  
            System.out.println(amount + " Deposited and Update Balance now is :" + balance);  
        } else {  
            System.out.println("Invalid Amount...");  
        }  
    }  
  
    public void withdraw(double amount) {  
        if(amount > 0 && amount <= balance) {  
            balance -= amount;  
            System.out.println(amount + " Withdrawn and Update Balance now is :" + balance);  
        } else {  
            System.out.println("Insufficient Balance...");  
        }  
    }  
}
```

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
balance -= amount;  
System.out.println(amount + " is deducted and remaining balance is: " + balance);  
} else {  
    System.out.println("Invalid amount or insufficient amount...");  
}  
}  
  
public double getBalance() {  
    return balance;  
}  
}
```

☞ Short Notes:

- ✓ Encapsulation ကို Java Beans တွေမှာ အများဆုံးအသုံးပြုတယ်
- ✓ Android Development မှာ Model Classes တွေရေးတဲ့အခါ အရေးကြီးတယ်
- ✓ Spring Framework တွေမှာ DTO (Data Transfer Objects) တွေရေးတဲ့အခါ အသုံးဝင်တယ်

Encapsulation သည် Java Programming မှာ အရေးကြီးတဲ့ concept တစ်ခုဖြစ်ပြီး professional level programming တွေအတွက် မရှိမဖြစ်လိုအပ်ပါတယ်။

▣ Inheritance

Inheritance (အမေဆက်ခံခြင်း) ဆိုတာ Java OOP ရဲ့ အရေးကြီးတဲ့ concept တစ်ခုဖြစ်ပြီး တစ်ခုထက်ပိုတဲ့ class တွေကြားမှာ parent-child relationship ဖန်တီးပေးတဲ့နည်းလမ်း ဖြစ်ပါတယ်။

□ Inheritance ရဲ့ အဓိက အချက်များ

1. Code Reusability - Parent class ၏ code တွေကို child class ၏ ပြန်သုံးလို့ရတယ်
2. Method Overriding - Child class ၏ parent class ရဲ့ method တွေကို modify လုပ်လို့ရတယ်
3. extends keyword - Inheritance လုပ်ဖို့အတွက် သုံးတယ်
4. IS-A Relationship - "Child IS-A Parent" ဆိုတဲ့ ဆက်နွယ်မှုကို ဖော်ပြုတယ်

□ Inheritance ကို ဘယ်လို အသုံးပြုလဲ?

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



1. Base Class (Parent Class) ဖန်တီးပါ

Code:

```
class Animal {  
    void eat() {  
        System.out.println("It is eating...");  
    }  
  
    void sleep() {  
        System.out.println("It is sleeping...");  
    }  
}
```

2.Derived Class (Child Class) ဖန်တီးပြီး extends လုပ်ပါ

Code:

```
class Dog extends Animal { // Dog သူ Animal မှာ inherit လုပ်တယ်  
    void bark() {  
        System.out.println( "It is barking now...");  
    }  
}
```

3.Main Class မှာ အသိုးပြုပါ

Code:

```
public class Main {  
    public static void main(String[] args) {  
        Dog myDog = new Dog();  
  
        // Parent class ကနေ inherit လုပ်ထားတဲ့ methods ဆွဲ  
        myDog.eat(); // Output: It is eating...  
        myDog.sleep(); // Output: It is sleeping...  
  
        // Dog class သူ method  
        myDog.bark(); // Output:It is barking now...  
    }  
}
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



❑ Inheritance အမျိုးအစားများ

1. Single Inheritance

Code:

```
class A {}  
class B extends A {} // B သဲ A ကိုယ် inherit လုပ်တယ်
```

1. Multilevel Inheritance

Code:

```
class A {}  
class B extends A {}  
class C extends B {} // C → B → A
```

2. Hierarchical Inheritance

Code:

```
class A {}  
class B extends A {}  
class C extends A {} // B နဲ့ C သဲ A ကိုယ် inherit လုပ်တယ်
```

Method Overriding လုပ်နည်း

Code:

```
class Animal {  
    void makeSound() {  
        System.out.println( "Animal Sound... ");  
    }  
}  
  
class Cat extends Animal {  
    @Override  
    void makeSound() { // Parent class သဲ method ကို override လုပ်တယ်  
        System.out.println( "Cat Sound... Meow!");  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {
```



```
Animal myCat = new Cat();
myCat.makeSound(); // Output: Cat Sound... Meow!
}
}
```

□ super keyword အသံးပြနည်း

Code:

```
class Vehicle {
    int speed = 50;
}

class Bike extends Vehicle {
    int speed = 100;

    void display() {
        System.out.println("Bike speed: " + speed);           // 100
        System.out.println("Vehicle speed: " + super.speed); // 50
    }
}
```

□ Inheritance ရဲအကိုးကျေးဇူးများ

1. Code Reuse - ထပ်ခါထပ်ခါရေးစရာမလိုဘေး
2. Method Overriding – Polymorphism အတွက်အရေးကြီးတယ်
3. Code Organization – Class တွက် hierarchy အလိုက်စီစဉ်တယ်

☞ Short Notes:

- ✓ Java မှာ Multiple Inheritance ကို class တွန့်မလုပ်နိုင်ဘူး (interface တွန့်ပဲလုပ်နိုင်တယ်)
- ✓ Constructor တွက် inherit မလုပ်နိုင်ဘူး
- ✓ Private members တွက် inherit မလုပ်နိုင်ဘူး

Inheritance က Java Programming မှာ အရေးကြီးတဲ့ concept တစ်ခုဖြစ်ပြီး real-world relationships တွက် program ထဲမှာ model လုပ်တဲ့အခါ အသံးဝင်ပါတယ်



Polymorphism

Polymorphism (တစ်ခုတည်းကို ပုံစံမျိုးစုံဖြင့် အသုံးပြုနိုင်ခြင်း) ဆိုတာ Java OOP ရဲ့ အရေးကြီးဆုံး concept တစ်ခုဖြစ်ပြီး “တစ်ခုတည်းသော interface ကို အမျိုးမျိုးသော implementation တွေနဲ့ အသုံးပြုနိုင်တဲ့ စွမ်းရည်” လို အဓိကဖွံ့ဖြိုးနိုင်ပါတယ်။

▣ Polymorphism ရဲ့ အဓိက အချက်များ

1. Method Overloading (Compile-time Polymorphism)
2. Method Overriding (Runtime Polymorphism)
3. Interface & Abstract Class တွေကို အသုံးပြုခြင်း

1. Method Overloading (Compile-time Polymorphism)

Code:

```
class Calculator {  
    // တူညီတဲ့ method name နဲ့ parameters မတူတဲ့ methods ထော်  
    int add(int a, int b) {  
        return a + b;  
    }  
  
    double add(double a, double b) {  
        return a + b;  
    }  
  
    String add(String a, String b) {  
        return a + b;  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Calculator calc = new Calculator();  
        System.out.println(calc.add(5, 10));      // 15
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
        System.out.println(calc.add(5.5, 10.5)); // 16.0
        System.out.println(calc.add("Hello", "World")); // HelloWorld
    }
}
```

2.Method Overriding (Runtime Polymorphism)

Code:

```
class Animal {
    void makeSound() {
        System.out.println( "Animal Sound...");}
}

class Cat extends Animal {
    @Override
    void makeSound() {
        System.out.println( "Cat Sound... Meow ... ");
    }
}

class Dog extends Animal {
    @Override
    void makeSound() {
        System.out.println( "Dog Barking ... Wok Wok...");}
}

public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Animal();
        Animal myCat = new Cat();
        Animal myDog = new Dog();

        myAnimal.makeSound(); // Animal Sound ...
        myCat.makeSound(); // Cat Sound ... Meow ...
        myDog.makeSound(); // Dog Barking ... Wok Wok...
    }
}
```



3. Interface Polymorphism

Code:

```
interface Shape {  
    void draw();  
}  
  
class Circle implements Shape {  
    @Override  
    public void draw() {  
        System.out.println("Drawing Circle Now...");  
    }  
}  
  
class Square implements Shape {  
    @Override  
    public void draw() {  
        System.out.println("Drawing Rectangle Now...");  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Shape myShape = new Circle();  
        myShape.draw(); // Drawing Circle Now...  
  
        myShape = new Square();  
        myShape.draw(); // Drawing Rectangle Now...  
    }  
}
```

▣ Polymorphism ရဲအကိုးကျေးဇူးများ

1. Code Flexibility - တစ်ခုတည်းသော interface ကို အမြိုးမျိုးသုံးနိုင်တယ်
2. Code Maintainability – Code ထွေကို ပြင်ဆင်ရတာ ပိုလွယ်ကူတယ်
3. Extensibility - အသစ်ထပ်ထည့်ရတာ လွယ်ကူတယ်



☞ Short Notes:

- ✓ Polymorphism ကို Java Collections Framework တွေမှာ အများဆုံးတွေ့ရတယ်
- ✓ Android Development မှာ Adapter Pattern တွေမှာ အသုံးများတယ်
- ✓ Spring Framework မှာ Dependency Injection လုပ်တဲ့အခါ အရေးပါတယ်

Polymorphism ကို Java Programming မှာ အရေးကြီးတဲ့ concept တစ်ခုဖြစ်ပြီး professional level programming တွေအတွက် မရှိမဖြစ်လိုအပ်ပါတယ်။



Abstraction

Abstraction ဆိတ် Java OOP ရဲ့အရေးကြီးတဲ့ concept တစ်ခုဖြစ်ပြီး "လိုအပ်တဲ့အချက်တွေကိုပြပြီး အတွင်းပိုင်း အသေးစိတ်တွေကို ဖုံးကွယ်ထားတဲ့ နည်းလမ်း" ဖြစ်ပါတယ်။ Abstraction ကို abstract class နဲ့ interface တွေသုံးပြီး implement လုပ်ပါတယ်။

□ Abstraction ရဲ့အဓိက အချက်များ

1. Essential features တွေကိုပြပါတယ်
2. Implementation details တွေကို ဖုံးကွယ်ထားတယ်
3. abstract keyword သုံးပြီး ဖန်တီးတယ်
4. 100% abstraction အတွက် interface တွေသုံးတယ်

⇒ Abstract Class အသုံးပြုနည်း

Code:

```
abstract class Vehicle {  
    // Abstract method (implementation မဟု)  
    abstract void start();  
  
    // Concrete method (implementation ဟု)  
    void stop() {  
        System.out.println( "stop the vehicle now...");  
    }  
}  
  
class Car extends Vehicle {  
    @Override  
    void start() {  
        System.out.println( "the car starts moving now...");  
    }  
}
```



```
class Bike extends Vehicle {  
    @Override  
    void start() {  
        System.out.println("the bike starts moving now...");  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Vehicle myCar = new Car();  
        Vehicle myBike = new Bike();  
  
        myCar.start(); // the car starts moving now...  
        myCar.stop(); // stop the vehicle now...  
  
        myBike.start(); // the bike starts moving now...  
        myBike.stop(); // stop the vehicle now...  
    }  
}
```

⇒ Interface အသုံးပြန်ညွှန် (100% Abstraction)

Code:

```
interface Bank {  
    // Abstract methods တွေပဲ  
    void deposit(double amount);  
    void withdraw(double amount);  
    double getBalance();  
}
```

```
class KBZBank implements Bank {  
    private double balance;  
  
    @Override  
    public void deposit(double amount) {  
        balance += amount;  
    }
```



```
}
```

```
    @Override
    public void withdraw(double amount) {
        if(balance >= amount) {
            balance -= amount;
        }
    }

    @Override
    public double getBalance() {
        return balance;
    }
}

public class Main {
    public static void main(String[] args) {
        Bank bank = new KBZBank();
        bank.deposit(500000);
        bank.withdraw(200000);
        System.out.println("The balance is: " + bank.getBalance());
    }
}
```

□ Abstraction ရဲအကိုးကျေးဇူးများ

1. Complexity ကိုလျှော့ချိန်တယ်
2. Code maintenance လုပ်ရတာလွယ်တယ်
3. Security ပိုကောင်းတယ်
4. Teamwork အတွက်အဆင်ပြတယ်



Abstract Class vs Interface

Feature	Abstract Class	Interface
Variables	Any type	Only public static final
Methods	Abstract + Concrete	Only abstract (Java 8+)
Inheritance	Single inheritance	Multiple inheritance
Constructor	ရှိ	မရှိ
Access Modifiers	Any	Only public

☞ Short Notes:

- ✓ Android Development မှ Adapter Pattern တွေမှာ အသုံးများတယ်
- ✓ Spring Framework မှ Service Layer တွေမှာ အရေးပါတယ်
- ✓ Java Collections Framework မှ List, Set, Map တွေမှာ အခြေခံအုတ်မြစ်ဖြစ်တယ်

Abstraction က Java Programming မှာ အရေးကြီးတဲ့ concept တစ်ခုဖြစ်ပြီး professional level programming တွေအတွက် မရှိမဖြစ်လိုအပ်ပါတယ်။



JAVA COLLECTIONS



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



Java Collections Framework မှာ Interface တွေက အရေးကြီးတဲ့ အခန်းကဏ္ဍမှာ ပါဝင်ပါတယ်။ Collection Interface တွေကို အသုံးပြုပြီး data structures တွေကို စနစ်တကျ စီမံနိုင်ပါတယ်။

□ Collections Framework ရဲ့အဓိက Interfaces

1. Collection<E> Interface

Collection သဲ root interface ဖြစ်ပြီး List, Set, Queue တို့အားလုံးက ဒီ interface ကို extend လုပ်ထားပါတယ်။
အဓိက method တွေ:

- add(E e),
- remove(Object o)
- size(),
- isEmpty()
- contains(Object o),
- clear()

2. List<E> Interface

List သဲ ordered collection ဖြစ်ပြီး duplicate values တွေကို လက်ခံပါတယ်။

အသုံးများတဲ့ Classes: ArrayList, LinkedList, Vector

အဓိက method တွေ:

- get(int index)
- set(int index, E element)
- indexOf(Object o)

3. Set<E> Interface

Set သဲ unique elements တွေကိုသာ သိမ်းပါတယ်။

အသုံးများတဲ့ Classes: HashSet, LinkedHashSet, TreeSet

အဓိက method တွေ:

- add(E e) (duplicate ဆိုရင် false return ပြန်မယ်)
- contains(Object o)

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



4. Queue<E> Interface

FIFO (First-In-First-Out) နည်းလမ်းနဲ့ အလုပ်လုပ်ပါတယ်။

အသုံးများတဲ့ Classes: LinkedList, PriorityQueue

အဓိက method တွေ:

- offer(E e) (element ထည့်စိုး)
- poll() (ပထမဆုံး element ကို ဖယ်ရှားစိုး)

5. Map<K, V> Interface

Map သဲ key-value pair တွေကို သိမ်းပါတယ်။

အသုံးများတဲ့ Classes: HashMap, LinkedHashMap, TreeMap

အဓိက method တွေ:

- put(K key, V value)
- get(Object key)
- containsKey(Object key)

ဥပမာ Code Snippet

Code:

```
import java.util.*;

public class CollectionsExample {
    public static void main(String[] args) {
        // List Example
        List<String> fruits = new ArrayList<>();
        fruits.add("Apple");
        fruits.add("Banana");
        System.out.println(fruits); // [Apple, Banana]

        // Set Example
        Set<Integer> numbers = new HashSet<>();
        numbers.add(10);
        numbers.add(20);
        numbers.add(10); // duplicate, not stored
        System.out.println(numbers); // [20, 10]

        // Map Example
    }
}
```



```
Map<String, Integer> ages = new HashMap<>();  
ages.put("Alice", 25);  
ages.put("Bob", 30);  
System.out.println(ages.get("Alice")); // 25  
}  
}
```

☞ Short Notes:

- List → Ordered, Duplicates allowed
- Set → No Duplicates
- Queue → FIFO နည်းလမ်း
- Map → Key-Value Pair

Java Collections Framework ကို သေချာနားလည်ထားရင် data တွေကို ထိနေရေးစွာ manage လုပ်နိုင်မှာ
ဖြစ်ပါတယ်။



LIST INTERFACE



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



LIST INTERFACE

List Interface က Collection Framework တဲ့မှာ အရေးကြီးတဲ့ အစိတ်အပိုင်းဖြစ်ပြီး ordered collection (အစဉ်လိုက်စိတ္တာသော အချက်အလက်စု) ကို ကိုယ်စားပြုပါ တယ်။ List က duplicate values တွေကို လက်ခံပြီး index-based နည်းလမ်းနဲ့ element တွေကို ထိန်းချုပ်နိုင်ပါတယ်။

□ List Interface ရဲ့အဓိက အချက်များ

- အစဉ်လိုက် သိမ်းဆည်းပေးမယ် (Insertion Order ကို ထိန်းသိမ်းပေးပါတယ်)
 - Duplicate တန်ဖိုးတွေကို ခွင့်ပြုမယ်
 - Index သုံးပြီး element တွေကို access လုပ်နိုင်မယ်
-

□ List Interface ကို Implement လုပ်ထားတဲ့ အသုံးများတဲ့ Classes

- ArrayList → Dynamic array ကို အခြေခံထားတယ်၊ အများဆုံးသုံးတယ်
 - LinkedList → Doubly-linked list ကို အခြေခံထားတယ်၊ frequent insertions/deletions အတွက် သင့်တော်တယ်
 - Vector → Thread-safe ဖြစ်တယ်၊ သို့သော် အသုံးနည်းတယ်
-

□ List Interface ရဲ့အဓိက Method များ

Method	ရှင်းလင်းချက်
add(E e)	List ရဲ့နောက်ဆုံးမှာ element ထည့်မယ်
add(int index, E e)	သတ်မှတ်ထားတဲ့ index မှာ element ထည့်မယ်
get(int index)	သတ်မှတ်ထားတဲ့ index က element ကို return ပြန်မယ်

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



Method	ရှင်းလင်းချက်
set(int index, E e)	သတ်မှတ်ထားတဲ့ index မှာရှိတဲ့ element ကို update လုပ်မယ်
remove(int index)	သတ်မှတ်ထားတဲ့ index က element ကို ဖျက်မယ်
size()	List ထဲက element အရေအတွက်ကို return ပြန်မယ်
contains(Object o)	List ထဲမှာ element ရှိမရှိ စစ်မယ်
clear()	List ထဲက element အားလုံးကို ဖျက်မယ်

ဥပမာ-

Code:

```
import java.util.ArrayList;
import java.util.List;

public class ListExample {
    public static void main(String[] args) {
        // ArrayList ကို List Interface နဲ့ Declare လုပ်မယ်
        List<String> fruits = new ArrayList<>();

        // add() method နဲ့ element ထွေထည့်မယ်
        fruits.add( "Apple" );
        fruits.add( "Banana" );
        fruits.add( "Mango" );
        fruits.add( "Banana" ); // Duplicate allowed

        System.out.println( "Initial List: " + fruits);
        // Output: [Apple, Banana, Mango, Banana]

        // get() method နဲ့ element ရယူမယ်
        String firstFruit = fruits.get(0);
        System.out.println( "First Fruit: " + firstFruit);
        // Output: Apple

        // set() method နဲ့ element ပြင်မယ်
        fruits.set(1, "Orange" );
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



```
System.out.println( "After Update: " + fruits);
// Output: [Apple, Orange, Mango, Banana]

// remove() method နဲ့ element ဖျက်မယ်
fruits.remove(2);
System.out.println( "After Removal: " + fruits);
// Output: [Apple, Orange, Banana]

// size() method နဲ့ list အရှယ်အစားစစ်မယ်
System.out.println( "List Size: " + fruits.size());
// Output: 3
}
```

▣ ArrayList vs LinkedList

Feature	ArrayList	LinkedList
အခြေခံ Data Structure	Dynamic Array	Doubly-Linked List
Access Speed (get)	ပိုမြန်တယ် ($O(1)$)	နေးတယ် ($O(n)$)
Insert/Delete Speed	နေးတယ် ($O(n)$)	ပိုမြန်တယ် ($O(1)$)
Memory Usage	နည်းတယ်	ဂိုလ်းတယ် (Node အတွက် extra memory)

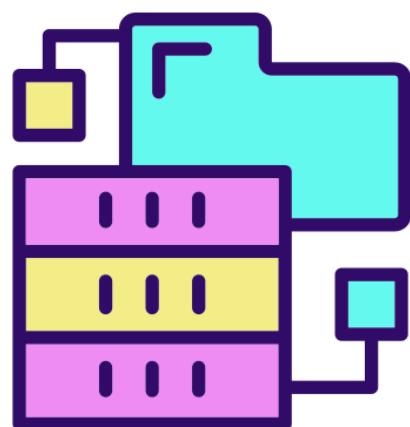
▣ ဘယ်အချင့်မှာ List ကိုသုံးမလဲ?

- ✓ ArrayList → အများဆုံးသုံးတယ်, random access လိုတဲ့အခါ (ဥပမာ: Database ကနေ data ဖတ်တဲ့အခါ)
- ✓ LinkedList → အကြိမ်ရောများများ add/remove လုပ်ရတဲ့အခါ (ဥပမာ: Real-time data processing)

List က Java Programming မှာ အရမ်းအသုံးဝင်တဲ့ Data Structure တစ်ခုဖြစ်ပြီး နားလည်ထားရင် Projects တွေမှာ ထိရောက်စွာ အသုံးချခိုင်ပါတယ်!



Set Interface



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



SET INTERFACE

Set Interface က Collection Framework ထဲမှာ အရေးကြီးတဲ့ အစိတ်အပိုင်းဖြစ်ပြီး unique elements only (ထူးခြားတဲ့ အချက်အလက်များသာ) ကို သိမ်းဆည်းပေး ပါတယ်။ Set က duplicate values တွေကို လုံးဝမလက်ခံပါဘူး။

Set Interface ရဲ့အဓိက အချက်များ

- Duplicate တန်ဖိုးတွေကို လုံးဝမလက်ခံဘူး
- Insertion Order ကို အာမမခံဘူး (အမျိုးအစားပေါ်မှုတည်ပါတယ်)
- null value ကို လက်ခံနိုင်တယ် (တစ်ခါတစ်ရုံ)

□ Set Interface ကို Implement လုပ်ထားတဲ့ အသုံးများတဲ့ Classes

- HashSet → Hash table ကို အခြေခံထားတယ်၊ အမြန်ဆုံး access ရပြီး order မရှိဘူး
- LinkedHashSet → Insertion order ကို ထိန်းသိမ်းပေးတယ်
- TreeSet → Red-Black tree ကို အခြေခံထားတယ်၊ sorted order နဲ့ပြပေးတယ်

□ Set Interface ရဲ့အဓိက Method များ

Method	ရှင်းလင်းချက်
add(E e)	Set ထဲကို element ထည့်မယ် (duplicate ဆိုရင် false return ပြန်မယ်)
remove(Object o)	Element ကို ဖျက်မယ်
contains(Object o)	Element ပါမလို စစ်မယ်
size()	Set ထဲက element အရေအတွက်
isEmpty()	Set က ဗလာလားစစ်မယ်
clear()	Set ထဲက element အားလုံးကို ဖျက်မယ်

ဥပမာ-

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



Code:

```
import java.util.HashSet;
import java.util.Set;

public class SetExample {
    public static void main(String[] args) {
        // HashSet ကို Set Interface နဲ့ Declare လုပ်မယ်
        Set<String> fruits = new HashSet<>();

        // add() method နဲ့ element တွေထည့်မယ်
        fruits.add("Apple");
        fruits.add("Banana");
        fruits.add("Mango");
        fruits.add("Apple"); // Duplicate ဖြစ်လို့ မထည့်ဘူး

        System.out.println("Initial Set: " + fruits);
        // Output: [Apple, Banana, Mango] (order မသေချာ)

        // contains() method နဲ့စစ်မယ်
        System.out.println("Contains Banana? " + fruits.contains("Banana"));
        // Output: true

        // remove() method နဲ့ဖျက်မယ်
        fruits.remove("Mango");
        System.out.println("After Removal: " + fruits);
        // Output: [Apple, Banana]

        // size() method နဲ့အရွယ်အစားစစ်မယ်
        System.out.println("Set Size: " + fruits.size());
        // Output: 2
    }
}
```

□ HashSet vs LinkedHashSet vs TreeSet

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



Feature	HashSet	LinkedHashSet	TreeSet
Ordering	No order	Insertion order	Sorted order
Performance	Fastest	Slightly slower	Slowest
null allowed	Yes	Yes	No
ဘယ်အချင့်သုံးမလဲ	Unique data အတွက်	Order ထိန်းဖို့လိုရင်	Sorted data လိုရင်

□ ဘယ်အချင့်မှာ Set ကိုသုံးမလဲ?

- ✓ ထူးခြားတဲ့တန်ဖိုးတွေသာ လိုအပ်တဲ့အခါ
- ✓ Duplicate ဖြစ်နေတာကို ရှောင်ချင်တဲ့အခါ
- ✓ အချက်အလက်တွေကို အမြန်ရှာဖွေဖို့လိုတဲ့အခါ

Set က Java Programming မှာ အရမ်းအသုံးဝင်တဲ့ Data Structure တစ်ခုဖြစ်ပြီး နားလည်ထားရင် Projects တွေမှာ ထိရောက်စွာ အသုံးချိန်ပါတယ်



MAP INTERFACE



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



MAP INTERFACE

Map Interface သည် key-value pairs (သော့နှင့်တန်ဖိုးစုံ) တွေကို သိမ်းဆည်းဖို့အတွက် အသုံးပြုပါတယ်။ Collection Framework ထဲမှာ အရေးကြီးတဲ့ အစိတ်အပိုင်း ဖြစ်ပြီး List နဲ့ Set တို့နဲ့ မတူညီတဲ့ အောက်ပါအချက်တွေရှိပါတယ်။

□ Map Interface ရဲ့အဓိက အချက်များ

1. Key-Value Pair အဖြစ် သိမ်းဆည်းမယ်
2. Key တွေက unique ဖြစ်ရမယ် (Duplicate keys မရှိဘူး)
3. Value တွေက duplicate ဖြစ်နိုင်တယ်
4. Key တစ်ခုကို null တစ်ခါသာ ထည့်နိုင်တယ် (HashMap/LinkedHashMap မှာ)
5. Value တွေကို null မကန့်သတ်ဘူး

□ Map Interface ကို Implement လုပ်ထားတဲ့ အသုံးများတဲ့ Classes

- HashMap → Hash table အခြေခံထားတယ်၊ အမြန်ဆုံး access ရပြီး order မရှိဘူး
- LinkedHashMap → Insertion order ကို ထိန်းသိမ်းပေးတယ်
- TreeMap → Red-Black tree အခြေခံထားတယ်၊ key အလိုက် sort လုပ်ပေးတယ်
- Hashtable → Thread-safe ဖြစ်တယ်၊ သို့သော် အသုံးနည်းတယ်

□ Map Interface ရဲ့အဓိက Method များ

Method	ရုပ်ငွေးလင်းချက်
put(K key, V value)	Map ထဲကို key-value pair ထည့်မယ်
get(Object key)	Key နဲ့သက်ဆိုင်တဲ့ value ကို return ပြန်မယ်
containsKey(Object key)	Key ရှိမရှိ စစ်မယ်
containsValue(Object value)	Value ရှိမရှိ စစ်မယ်
remove(Object key)	Key နဲ့သက်ဆိုင်တဲ့ entry ကို ဖျက်မယ်
size()	Map ထဲက entry အရေအတွက်

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



Method	ရှင်းလင်းချက်
keySet()	Key တွေအားလုံးကို Set အဖြစ် return ပြန်မယ်
values()	Value တွေအားလုံးကို Collection အဖြစ် return ပြန်မယ်
entrySet()	Key-value pairs အားလုံးကို Set အဖြစ် return ပြန်မယ်

ဥပမာ-

Code:

```
import java.util.HashMap;
import java.util.Map;

public class MapExample {
    public static void main(String[] args) {
        // HashMap ကို Map Interface နဲ့ Declare လုပ်မယ်
        Map<String, Integer> ageMap = new HashMap<>();

        // put() method နဲ့ entry တွေထည့်မယ်
        ageMap.put("Alice", 25);
        ageMap.put("Bob", 30);
        ageMap.put("Charlie", 35);
        ageMap.put("Alice", 26); // Duplicate key - value ကို update လုပ်မယ်

        System.out.println("Initial Map: " + ageMap);
        // Output: {Alice=26, Bob=30, Charlie=35}

        // get() method နဲ့ value ရယူမယ်
        int aliceAge = ageMap.get("Alice");
        System.out.println("Alice's Age: " + aliceAge);
        // Output: 26

        // containsKey() method နဲ့ စစ်မယ်
        System.out.println("Contains key 'Bob'? " + ageMap.containsKey("Bob"));
        // Output: true

        // remove() method နဲ့ entry ဖျက်မယ်
        ageMap.remove("Charlie");
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
System.out.println("After Removal: " + ageMap);
// Output: {Alice=26, Bob=30}

// size() method နဲ့အရွယ်အစားစစ်မယ်
System.out.println("Map Size: " + ageMap.size());
// Output: 2
}
}
```

□ HashMap vs LinkedHashMap vs TreeMap

Feature	HashMap	LinkedHashMap	TreeMap
Ordering	No order	Insertion order	Sorted by keys
Performance	Fastest	Slightly slower	Slowest
null keys	1 allowed	1 allowed	Not allowed
null values	Allowed	Allowed	Allowed
ဘယ်အချင့်သုံးမလဲ	General use	Insertion order လိုဂ်	Sorted keys လိုဂ်

□ ဘယ်အချင့်မှာ Map ကိုသုံးမလဲ?

- ✓ Key နဲ့ Value ဆက်စပ်မှုလိုတဲ့အခါ
- ✓ အချက်အလက်တွေကို key အရ အမြန်ရှာဖွေဖို့လိုတဲ့အခါ
- ✓ ဒေတာတွေကို အမျိုးအစားခွဲသိမ်းဖို့လိုတဲ့အခါ

Map က Java Programming မှာ အရမ်းအသုံးဝင်တဲ့ Data Structure တစ်ခုဖြစ်ပြီး နားလည်ထားရင် Projects တွေမှာ ထိရောက်စွာ အသုံးချိန်ပါတယ!



LAMBDA



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။

 **LAMBDA**

Java Lambda Expression သည် Java 8 မှစတင်မိတ်ဆက်ခဲ့တဲ့ အောင်ရပ်တစ်ခုဖြစ်ပြီး functional programming ကို ပိုမိုလွယ်ကူစွာ ရေးသားနိုင်ဖို့ ကူညီပေးပါတယ်။

□ Lambda ဆိတာဘာလဲ?

Lambda သည် anonymous function (အမည်မဲ့ function) တစ်ခုဖြစ်ပြီး အောက်ပါအချက်တွေကိုလုပ်ဆောင်နိုင်ပါတယ်:

- Method ကိုပါမယ့် parameter list တစ်ခုကို လက်ခံနိုင်ပါတယ်
- Body တစ်ခုရှုပါတယ်
- Return type ရှိပါတယ် (ဒါပေမယ့် မကြညာဘဲထားနိုင်ပါတယ်)
- throws exception ပါနိုင်ပါတယ်

□ Lambda Syntax

Lambda expression ၏ အခြေခံပံ့ဌံး :

Code:

(parameters) -> expression

သို့မဟုတ်

Code:

(parameters) -> { statements; }



⇒ ဥပမာများ

1. ရှိခိုး Lambda

Code:

```
() -> System.out.println("Hello Lambda!");
```

2. Parameter တစ်ခုပါ Lambda

Code:

```
(name) -> System.out.println("Hello " + name)
```

3. Multiple Parameters

Code:

```
(a, b) -> a + b
```

4. Multiple Statements

Code:

```
(name) -> {
    String greeting = "Hello " + name;
    System.out.println(greeting);
    return greeting;
}
```

Lambda ကိုဘယ်မှာသုံးမလဲ?

Lambda expression တွေကို အထူးသဖြင့် functional interface တွေနဲ့ တွဲသုံးပါတယ်။ Functional interface ဆိုတာက method တစ်ခုပဲပါတဲ့ interface ဖြစ်ပါတယ်။

Runnable Interface နဲ့သုံးတဲ့ဥပမာ

Code:

```
// ရေးရှိနည်း
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



```
Runnable r1 = new Runnable() {  
    @Override  
    public void run() {  
        System.out.println("Hello World 1");  
    }  
};  
  
// Lambda သုံးပြီးရေးနည်း  
Runnable r2 = () -> System.out.println("Hello World 2");
```

Comparator Interface နှင့်ဥပမာ

Code:

```
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
```

```
// ရေးရှိနည်း  
Collections.sort(names, new Comparator<String>() {  
    @Override  
    public int compare(String a, String b) {  
        return a.compareTo(b);  
    }  
});
```

```
// Lambda သုံးပြီးရေးနည်း  
Collections.sort(names, (a, b) -> a.compareTo(b));
```

✓ Lambda ခဲ့အားသာချက်များ

1. Code ကိုတို့တောင်းစေပါတယ် - Anonymous class တွေထက် ပိုတို့တောင်းပါတယ်
2. ဖတ်ရလွယ်ကူစေပါတယ် - code ကိုပိုပြီးရှင်းလင်းစွာမြင်နိုင်ပါတယ်
3. Functional programming ကိုအားပေးပါတယ် - Java မှာ functional style နှင့်ရေးနိုင်ပါတယ်

□ Method References

Lambda နှင့်ဆင်တဲ့အခြားအကိုဂျင်တစ်ခုမှာ method reference ဖြစ်ပါတယ်။

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



ဥပမာ:

Code:

```
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
```

// Lambda သုံးပြီးရေးနည်း

```
names.forEach(name -> System.out.println(name));
```

// Method reference သုံးပြီးရေးနည်း

```
names.forEach(System.out::println);
```

Lambda expression များသည် Java programming ကို ပိုမိုတိုးတက်စေပြီး ကုဒ်ရေးသားမှုကို ပိုမိုလွယ်ကူပေါ်တယ်။

❑ Java Lambda အသုံးပြု၍ ပေါင်းခြင်းနှင့်နှုတ်ခြင်း

1. Functional Interface သတ်မှတ်ခြင်း

ပထားဆုံး အပေါင်း အနှုတ်လုပ်ဆောင်ချက်များအတွက် functional interface တစ်ခုကိုသတ်မှတ်ပါမယ်။

Code:

```
@FunctionalInterface  
interface MathOperation {  
    int operate(int a, int b);  
}
```

2. Lambda Expressions အသုံးပြုခြင်း

Code:

```
public class LambdaMathExample {  
    public static void main(String[] args) {  
        // Lambda expression အသုံးပြု၍ ပေါင်းခြင်း  
        MathOperation addition = (a, b) -> a + b;  
  
        // Lambda expression အသုံးပြု၍ နှုတ်ခြင်း  
        MathOperation subtraction = (a, b) -> a - b;
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
// နမူနာတန်ဖိုးများ  
int num1 = 10;  
int num2 = 5;  
  
// ပေါင်းခြင်းကိုအသုံးပြုခြင်း  
System.out.println(num1 + " + " + num2 + " = " + addition.operate(num1, num2));  
  
// နှုတ်ခြင်းကိုအသုံးပြုခြင်း  
System.out.println(num1 + " - " + num2 + " = " + subtraction.operate(num1, num2));  
}  
}
```

3. Output ရလဒ်

အထက်ပါကြိုင်ကို run ပါက အောက်ပါအတိုင်းရလဒ်ထွက်ပါမယ်:

Console:

10 + 5 = 15

10 - 5 = 5

4. နောက်ထပ်နည်းလမ်း - Method Parameter အဖြစ် Lambda ကိုသုံးခြင်း

Code:

```
public class LambdaMathExample2 {  
    public static void main(String[] args) {  
        int num1 = 20;  
        int num2 = 8;  
  
        // Lambda expression ကို တိုက်ရှိကုသုံးခြင်း  
        System.out.println(num1 + " + " + num2 + " = " + operate(num1, num2, (a, b) -> a + b));  
        System.out.println(num1 + " - " + num2 + " = " + operate(num1, num2, (a, b) -> a - b));  
    }  
  
    public static int operate(int a, int b, MathOperation operation) {  
        return operation.operate(a, b);  
    }  
}
```

5. ရှင်းလင်းချက်

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



1. MathOperation သည် functional interface တစ်ခုဖြစ်ပြီး operate ဆိုတဲ့ method တစ်ခုပါရှိပါတယ်။
2. addition နှင့် subtraction တို့သည် lambda expression များဖြစ်ကြပါတယ်။
3. Lambda expression များသည် interface တစ်ခုရဲ့ method ကိုအကောင်အထည်ဖော်ပေးပါတယ်။
4. ဒီနည်းလမ်းဖြင့် ကုဒ်များကိုပိမိတို့တောင်းစွာရေးသားနိုင်ပြီး ဖတ်ရှုရလွယ်ကူပါတယ်။

Lambda expression များသည် Java တွင် functional programming ကိုအသုံးပြုရာတွင် အလွန်အသုံးဝင်ပါတယ်။



လေ့ကျင့်ရန် -

Java Lambda အသုံးပြု၍ မြောက်ခြင်း နဲ့ စားခြင်းကို လေ့ကျင့်ပါ။

Output:

$10 * 5 = 150$

$10 / 2 = 5.0$



DATE



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



DATE

Java မှာ Date နဲ့ Time ကို ကိုင်တွယ်ဖို့ အဓိက Class ၏ မျိုးရှိပါတယ်။

1. java.util.Date (Legacy)

Code:

```
import java.util.Date;
```

```
public class Main {  
    public static void main(String[] args) {  
        Date currentDate = new Date();  
        System.out.println("လက်ရှိအချိန်: " + currentDate);  
    }  
}
```

▣ အသံးပြုပဲ

ရှိုးရှိုးရှင်းရှင်း Date နဲ့ Time ကိုပြချင်တဲ့ အခါ

ဒါပေမယ့် ဒီ Class ၏ Deprecated ဖြစ်နေပြီး မသုံးသင့်တော့ပါ။

2. java.util.Calendar (Legacy)

Code:

```
import java.util.Calendar;
```

```
public class Main {  
    public static void main(String[] args) {  
        Calendar cal = Calendar.getInstance();  
        int year = cal.get(Calendar.YEAR);  
        int month = cal.get(Calendar.MONTH) + 1; // 0-based  
        int day = cal.get(Calendar.DAY_OF_MONTH);  
  
        System.out.println("ယနေ့ရက်စွဲ: " + year + "-" + month + "-" + day);  
    }  
}
```

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



}

□ အသုံးပြုပုံ:

⇒ Date တွေကို Manipulate လုပ်ချင်တဲ့အခါ Calendar ကိုသုံးပါတယ်။

3. java.time Package (Modern - Java 8+)

Java 8 မှာ ပိုကောင်းတဲ့ Date/Time API တွေ မိတ်ဆက်ပေးလိုက်ပါတယ်။

⇒ LocalDate (နေ့စွဲသာ)

```
import java.time.LocalDate;
```

```
public class Main {  
    public static void main(String[] args) {  
        LocalDate today = LocalDate.now();  
        System.out.println("ယနေ့ရက်စွဲ: " + today);  
  
        LocalDate specificDate = LocalDate.of(2023, 12, 25);  
        System.out.println("ခရစ္စမတ်နေ့: " + specificDate);  
    }  
}
```

⇒ LocalTime (အချိန်သာ)

```
import java.time.LocalTime;
```

```
public class Main {  
    public static void main(String[] args) {  
        LocalTime now = LocalTime.now();  
        System.out.println("လက်ရှိအချိန်: " + now);  
    }  
}
```

⇒ LocalDateTime (နေ့စွဲအချိန်)

■ မြန်မာဖြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



```
import java.time.LocalDateTime;

public class Main {
    public static void main(String[] args) {
        LocalDateTime current = LocalDateTime.now();
        System.out.println("လက်ရှိ: " + current);
    }
}
```

⇒ DateTimeFormatter (ပုံစံပြောင်း)

```
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Main {
    public static void main(String[] args) {
        LocalDateTime now = LocalDateTime.now();
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
        String formattedDate = now.format(formatter);
        System.out.println("ဖော်မတ်ကျသောရက်စွဲ: " + formattedDate);
    }
}
```

□ Short Notes

- ✓ Java 8+ သုံးရင် java.time package ကိုသုံးပါ
- ✓ Legacy Code (Date, Calendar) ကို ရောင်ပါ
- ✓ Formatting လုပ်ချင်ရင် DateTimeFormatter သုံးပါ
- ✓ Time Zone လိုရင် ZonedDateTime သုံးပါ



REGULAR EXPRESSION



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



REGULAR EXPRESSION

Regular Expression (Regex) ဆိုတာ စာသားပုံစံများကို ရှာဖွေရန်၊ စစ်ဆေးရန်နှင့် ပြုပြင်ရန် အသုံးပြုတဲ့ စံသတ်မှတ်ချက်တစ်ခုဖြစ်ပါတယ်။ Java မှာ java.util.regex package ကို အသုံးပြုပြီး Regex များကို အလွယ်တကူ အသုံးပြုနိုင်ပါတယ်။

1. Java Regex အခြေခံများ

□ Pattern Class နှင့် Matcher Class

Code:

```
import java.util.regex.*;
```

```
String text = "Java Programming 2023";
Pattern pattern = Pattern.compile("\d+"); // ဂဏန်းများရှာမယ်
Matcher matcher = pattern.matcher(text);
```

```
while (matcher.find()) {
    System.out.println("တွေ့ရှိခဲ့သည်: " + matcher.group());
}
// ရလဒ်: တွေ့ရှိခဲ့သည်: 2023
```

String Class မှာ တိုက်ရှိက်သုံးနည်း

Code:

```
String email = "user@example.com" ;
boolean isValid = email.matches( "^[\\w.-]+@[\\w.-]+\\.\\w{2,6}$" );
System.out.println( "Email မှန်ကော်မူ: " + isValid);
// ရလဒ်: Email မှန်ကော်မူ: true
```

2. အသုံးများသော Regex Patterns

Pattern	ဖော်ပြချက်	ဥပမာ
\d	ဂဏန်းတစ်လုံး (0-9)	"Java8" → 8
\D	ဂဏန်းမဟုတ်သော အကွဲရာ	"Java8" → J,a,v,a
\w	Word Character (a-z, A-Z, 0-9, _)	"user_name123" → u,s,e,r,_n,a,m,e,1,2,3

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



Pattern	ဖော်ပြချက်	ဥပမာ
\W	Word Character မဟုတ်သော အကွဲရာ	"a@b#c" → @,#
\s	White Space (space, tab, newline)	"a b c" → (space)
\S	White Space မဟုတ်သော အကွဲရာ	"a b c" → a,b,c
[abc]	a, b သို့မဟုတ် c	"apple" → a,p,p
[^abc]	a, b, c မဟုတ်သော အကွဲရာ	"apple" → l,e
^pattern	စာကြောင်းအစဉ်နှင့် ကိုက်ညီမှု	^Java → "Java" (အစမှာရှိမယ်)
pattern\$	စာကြောင်းအဆုံးနှင့် ကိုက်ညီမှု	Script\$ → "JavaScript" (အဆုံးမှာရှိမယ်)

3. အသုံးဝင်သော Regex ဥပမာများ

⇒ Email Validation

Code:

```
String emailRegex = "^[\\w.-]+@[\\w.-]+\\.\\.[a-z]{2,6}$" ;
String email = "test.email+2023@gmail.com" ;
System.out.println(email.matches(emailRegex)); // true
```

⇒ Phone Number Validation (မြန်မာဖုန်းနံပါတ်)

Code:

```
String phoneRegex = "^(09|\\+?959)\\d{9}$";
String phone = "09123456789";
System.out.println(phone.matches(phoneRegex)); // true
```

⇒ Password Strength Checker

Code:

```
// အနည်းဆုံး 8 လုံး၊ စာကြီး၊ စာသေး၊ ဂဏ်း၊ အထူးသက်တဲ့ တစ်ခုပါရမယ်
String passwordRegex = "^(?=.*[A-Z])(?=.*[a-z])(?=.*[\\d])(?=.*[@#$%^&+=]).{8,$}" ;
String password = "Passw0rd#" ;
System.out.println(password.matches(passwordRegex)); // true
```

⇒ HTML Tag ဖယ်ရှားခြင်း

Code:

```
String html = "<p>Hello <b>World</b></p>" ;
String cleanText = html.replaceAll("<[^>]*>", "");
```



```
System.out.println(cleanText); // Hello World
```

⇒ Grouping & Capturing

Code:

```
String dateText = "2023-12-31" ;
Pattern datePattern = Pattern.compile( "(\\d{4})-(\\d{2})-(\\d{2})" );
Matcher dateMatcher = datePattern.matcher(dateText);

if (dateMatcher.matches()) {
    System.out.println( "နှစ်: " + dateMatcher.group(1)); // 2023
    System.out.println( "လ: " + dateMatcher.group(2)); // 12
    System.out.println( "ရက်: " + dateMatcher.group(3)); // 31
}
```

5. Regex Flags

Flag	အဓိပ္ပာယ်
Pattern.CASE_INSENSITIVE	အကြီးအသေး မခွဲခြားဘဲရှာမယ်
Pattern.MULTILINE	Multi-line mode တွင်သုံးမယ်
Pattern.DOTALL	Newline အပါအဝင် ကိုက်ညီမှုရှာမယ်

Code:

```
Pattern.compile("java", Pattern.CASE_INSENSITIVE).matcher("JAVA").find(); // true
```



□ Short Notes

1. Java မှာ \\ ကို Escape Character အဖြစ်သုံးရပါတယ် (ဥပမာ - \\d ဆိုရင် \d ကိုဆိုလိုတာပါ)
2. Pattern နဲ့ Matcher ကို ကြိမ်ဖန်များစွာအသုံးပြုမယ်ဆိုရင် Pattern.compile() ကို အရင်လုပ်ပါ
3. String.matches() method က တစ်ခုလုံးကိုက်ညီမှ true ပြန်ပေးပါတယ်

Java Regex ကို ကျမ်းကျင်စွာအသုံးပြုနိုင်ရင် Text Processing လုပ်ငန်းတွေကို အထူးလွယ်ကူမြန်ဆနွေးလုပ်ဆောင်နိုင်ပါတယ်!



RANDOM



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။

 RANDOM

□ Random Number

⇒ java.util.Random Class အသုံးပြုခြင်း

Code:

```
import java.util.Random;
```

```
Random random = new Random();
```

// 0 မှ 99 အထိ ကျပန်းကဏ္ဍား

```
int randomNumber = random.nextInt(100);
System.out.println("ကျပန်းကဏ္ဍား: " + randomNumber);
```

// 1 မှ 6 အထိ (အန်စာတံ့လို)

```
int diceRoll = random.nextInt(6) + 1;
System.out.println("အန်စာတံ့လိုမှုမြှင့်ခြင်း: " + diceRoll);
```

// ကျပန်း Boolean တန်ဖိုး

```
boolean randomBool = random.nextBoolean();
System.out.println("ကျပန်း Boolean: " + randomBool);
```

⇒ Math.random() Method အသုံးပြုခြင်း

Code:

// 0.0 (အပါအဝင်) မှ 1.0 (မပါဝင်) အထိ

```
double randomDouble = Math.random();
System.out.println("Math.random() ရလဒ်: " + randomDouble);
```

// 1 မှ 100 အထိ ပြောင်းလဲခြင်း

```
int randomInt = (int)(Math.random() * 100) + 1;
System.out.println("1-100 အတွင်းကျပန်းကဏ္ဍား: " + randomInt);
```

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူဗာ တတ်ရမည်။



⇒ Java 8+ ThreadLocalRandom Class

Code:

```
import java.util.concurrent.ThreadLocalRandom;
```

```
// ပိုမိုကောင်းမွန်သော နည်းလမ်း (Multi-thread အခြေအနေများအတွက်)
```

```
int randomNum = ThreadLocalRandom.current().nextInt(1, 101); // 1-100  
System.out.println("ThreadLocalRandom: " + randomNum);
```

□ Random String

⇒ Basic Method

Code:

```
String chars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";  
Random random = new Random();  
StringBuilder sb = new StringBuilder();
```

```
for (int i = 0; i < 10; i++) {  
    int index = random.nextInt(chars.length());  
    sb.append(chars.charAt(index));  
}
```

```
String randomString = sb.toString();  
System.out.println("ကျပ်နှီးစာသား: " + randomString);
```

⇒ Java 8 Stream အသုံးပြုခြင်း

Code:

```
import java.util.stream.Collectors;  
import java.util.stream.IntStream;
```

```
String randomStr = IntStream.range(0, 8)  
.mapToObj(i -> random.nextInt(36) < 26 ?  
    String.valueOf((char)(random.nextInt(26) + 'a')) :  
    String.valueOf(random.nextInt(10)))
```



```
.collect(Collectors.joining());
```

```
System.out.println("Stream ဖွင့်ကျပန်းစာသား: " + randomStr);
```

⇒ UUID အသုံးပြုခြင်း

Code:

```
import java.util.UUID;
```

```
String uuid = UUID.randomUUID().toString();
System.out.println("UUID: " + uuid);
// ဥပမာ: 550e8400-e29b-41d4-a716-446655440000
```

3. အထူးကျပန်းစာသားများထုတ်ယူနည်း

⇒ စာလုံးကြီးများသာ

Code:

```
String upperCaseRandom = random.ints(65, 91)
    .limit(8)
    .collect(StringBuilder::new, StringBuilder::appendCodePoint, StringBuilder::append)
    .toString();
```

```
System.out.println("စာလုံးကြီးများသား: " + upperCaseRandom);
```

⇒ ဂဏန်းများသာ

Code:

```
String numericRandom = random.ints(48, 58)
    .limit(6)
    .collect(StringBuilder::new, StringBuilder::appendCodePoint, StringBuilder::append)
    .toString();
```

```
System.out.println("ဂဏန်းများသား: " + numericRandom);
```



⇒ အထူးသက်တများပါဝင်သော

Code:

```
String specialChars = "!@#$%^&*()_+";
String allChars = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789" +
specialChars;
```

```
String complexRandom = random.ints(0, allChars.length())
    .limit(12)
    .map(allChars::charAt)
    .collect(StringBuilder::new, StringBuilder::appendCodePoint, StringBuilder::append)
    .toString();
```

```
System.out.println("အထူးစာသား: " + complexRandom);
```

4. အသုံးဝင်သော နမူနာများ

⇒ လုပ်ချော်အတွက် ကျပန်းစာသား (Password Generator)

Code:

```
public static String generateSecurePassword(int length) {
    String uppercase = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" ;
    String lowercase = "abcdefghijklmnopqrstuvwxyz" ;
    String numbers = "0123456789" ;
    String specialChars = "!@#$%^&*()_+" ;
    String combined = uppercase + lowercase + numbers + specialChars;
```

```
Random random = new Random();
StringBuilder sb = new StringBuilder();
```

```
for (int i = 0; i < length; i++) {
    int index = random.nextInt(combined.length());
    sb.append(combined.charAt(index));
}
```



```
return sb.toString();
}
```

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူဗာ တတ်ရမည်။



```
System.out.println("လုပ်ချောင်း: " + generateSecurePassword(12));
```

⇒ OTP (One-Time Password) ထုတ်ယူခြင်း

Code:

```
public static String generateOTP(int length) {  
    String numbers = "0123456789";  
    Random random = new Random();  
    StringBuilder sb = new StringBuilder();  
  
    for (int i = 0; i < length; i++) {  
        sb.append(numbers.charAt(random.nextInt(numbers.length())));  
    }  
  
    return sb.toString();  
}
```

```
System.out.println("OTP နံပါတ်: " + generateOTP(6));
```

□ Short Notes

1. `java.util.Random` သည် pseudo-random ဖြစ်ပါတယ်
(တစ်ခါတစ်ရုံတူညီတဲ့အစီအစဉ်အတိုင်းထုတ်ပေးနိုင်ပါတယ်)
2. လုပ်ချောင်းတဲ့အခါမျိုးမှာ `java.security.SecureRandom` ကိုသုံးပါ
3. `Math.random()` သည် double တန်ဖိုးပတ်ပေးပါတယ်
4. Multi-threaded application တွေမှာ `ThreadLocalRandom` ကိုသုံးပါ

Java မှာ ကျပန်းဂဏန်းနဲ့စာသားများကို အလွယ်တကူထုတ်ယူနိုင်ပြီး အမျိုးမျိုးသော အခြေအနေများအတွက် သင့်တော်စွာ အသုံးပြနိုင်ပါတယ်!



FILE HANDLING



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



FILE HANDLING

File နဲ့အလုပ်လုပ်ဖို့ java.io package ထဲက class တွေကိုအသုံးပြုပါတယ်။ File တွေကို ဖတ်ခြင်း၊ ရေးခြင်း၊ ဖျက်ခြင်းနဲ့ အခြား file operations တွဲလုပ်ဆောင်နိုင်ပါတယ်။

□ File Class အဓိက အချက်များ

1. File object တစ်ခုက file သို့မဟုတ် directory path ကိုကိုယ်စားပြုပါတယ်
2. File တည်ရှိမရှိ စစ်ဆေးနိုင်တယ်
3. File အချက်အလက်များ (အရွယ်အစား၊ နောက်ဆုံးပြင်ဆင်မှုအခါန်) ရယူနိုင်တယ်
4. File ဖန်တီးခြင်း၊ ဖျက်ခြင်း၊ အမည်ပြောင်းခြင်း လုပ်ဆောင်နိုင်တယ်

□ အဓိက File Operations

1. File ဖန်တီးခြင်း

Code:

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

```
public class FileExample {  
    public static void main(String[] args) {  
        try {  
            File myFile = new File("example.txt");  
            if (myFile.createNewFile()) {  
                System.out.println("File created: " + myFile.getName());  
            } else {  
                System.out.println("File already exists.");  
            }  
        } catch (IOException e) {  
            System.out.println("An error occurred.");  
            e.printStackTrace();  
        }  
    }  
}
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
}
```

2. File အချက်အလက်များရယူခြင်း

Code:

```
File myFile = new File("example.txt");
if (myFile.exists()) {
    System.out.println("File name: " + myFile.getName());
    System.out.println("Absolute path: " + myFile.getAbsolutePath());
    System.out.println("File size (bytes): " + myFile.length());
    System.out.println("Last modified: " + new Date(myFile.lastModified()));
} else {
    System.out.println("File does not exist.");
}
```

3. File ဖတ်ခြင်းနဲ့ရေးခြင်း

File ရေးခြင်း (FileWriter အသုံးပြုခြင်း)

Code:

```
import java.io.FileWriter;
try (FileWriter writer = new FileWriter("example.txt")) {
    writer.write("Hello Java File Handling!");
    System.out.println("Successfully wrote to the file.");
} catch (IOException e) {
    System.out.println("An error occurred.");
    e.printStackTrace();
}
```

File ဖတ်ခြင်း (Scanner အသုံးပြုခြင်း)

Code:

```
import java.io.File;
import java.util.Scanner;
try {
    File myFile = new File("example.txt");
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
Scanner myReader = new Scanner(myFile);
while (myReader.hasNextLine()) {
    String data = myReader.nextLine();
    System.out.println(data);
}
myReader.close();
} catch (FileNotFoundException e) {
    System.out.println("File not found.");
    e.printStackTrace();
}
```

❑ Directory အလုပ်လုပ်ခြင်း

Directory ဖွန်စီးခြင်း

Code:

```
File dir = new File("myDirectory");
if (dir.mkdir()) {
    System.out.println("Directory created successfully");
} else {
    System.out.println("Failed to create directory");
}
```

Directory ဆဲ့ File များစာရင်းရယူခြင်း

Code:

```
File dir = new File("myDirectory");
String[] files = dir.list();
for (String file : files) {
    System.out.println(file);
}
```



□ သတိပြုရန်အချက်များ

1. File operations တွေမှာ IOException ကို အမြဲ handle လုပ်ပါ
2. Resources တွေကို try-with-resources နဲ့ အသုံးပြုပါ (Java 7 နှင့်အထက်)
3. File path တွေမှာ absolute path နဲ့ relative path ကိုသတိထားပါ

□ ဘယ်အချိန်မှာ File Handling ကိုသုံးမလဲ?

- ✓ Configuration files တွေဖတ်ဖို့
- ✓ အချက်အလက်များ သိမ်းဆည်းဖို့
- ✓ Log files တွေရေးဖို့
- ✓ အချက်အလက်များ import/export လုပ်ဖို့

File Handling ကို နားလည်ထားရင် သင့် program တွေကို ပိုမိုအဆင့်မြင့်စွာ ရေးသားနိုင်မယ်ဖြစ်ပါတယ!



လေ့ကျင့်ရှင်-

- ✓ Java Console Post CRUD Application

အောက်ပါအတိုင်း Java console-based Post CRUD application တစ်ခုကို ရေးပြုမယ်။

Project Structure

```
post-crud-app/  
└── src/  
    ├── model/  
    │   └── Post.java  
    ├── repository/  
    │   └── PostRepository.java  
    │   └── FilePostRepository.java  
    └── service/
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
|  |  └ PostService.java  
|  └ MainApp.java  
└ posts_data.txt
```

1. Model Class - Post.java

Code:

```
package model;  
  
public class Post {  
    private int id;  
    private String title;  
    private String description;  
  
    public Post(int id, String title, String description) {  
        this.id = id;  
        this.title = title;  
        this.description = description;  
    }  
  
    // Getters and Setters  
    public int getId() {  
        return id;  
    }  
  
    public void setId(int id) {  
        this.id = id;  
    }  
  
    public String getTitle() {  
        return title;  
    }  
  
    public void setTitle(String title) {  
        this.title = title;  
    }  
  
    public String getDescription() {
```



```
        return description;
    }

    public void setDescription(String description) {
        this.description = description;
    }

    @Override
    public String toString() {
        return "ID: " + id + ", Title: " + title + ", Description: " + description;
    }
}
```

2. Repository Interface - PostRepository.java

Code:

```
package repository;

import model.Post;
import java.util.List;

public interface PostRepository {
    List<Post> getAllPosts();
    Post getPostById(int id);
    void addPost(Post post);
    void updatePost(Post post);
    void deletePost(int id);
}
```

3. File Implementation - FilePostRepository.java

Code:

```
package repository;

import model.Post;
import java.io.*;
import java.util.ArrayList;
import java.util.List;
```



```
public class FilePostRepository implements PostRepository {  
    private static final String FILE_PATH = "posts_data.txt";  
  
    @Override  
    public List<Post> getAllPosts() {  
        List<Post> posts = new ArrayList<>();  
        try (BufferedReader reader = new BufferedReader(new FileReader(FILE_PATH))) {  
            String line;  
            while ((line = reader.readLine()) != null) {  
                String[] parts = line.split("\\|");  
                if (parts.length == 3) {  
                    int id = Integer.parseInt(parts[0]);  
                    posts.add(new Post(id, parts[1], parts[2]));  
                }  
            }  
        } catch (IOException e) {  
            System.err.println("Error reading file: " + e.getMessage());  
        }  
        return posts;  
    }  
  
    @Override  
    public Post getPostById(int id) {  
        return getAllPosts().stream()  
            .filter(post -> post.getId() == id)  
            .findFirst()  
            .orElse(null);  
    }  
  
    @Override  
    public void addPost(Post post) {  
        try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_PATH, true))) {  
            writer.write(post.getId() + "|" + post.getTitle() + "|" + post.getDescription());  
            writer.newLine();  
        } catch (IOException e) {  
            System.err.println("Error writing to file: " + e.getMessage());  
        }  
    }  
}
```



```
@Override  
public void updatePost(Post updatedPost) {  
    List<Post> posts = getAllPosts();  
    for (int i = 0; i < posts.size(); i++) {  
        if (posts.get(i).getId() == updatedPost.getId()) {  
            posts.set(i, updatedPost);  
            break;  
        }  
    }  
    saveAllPosts(posts);  
}  
  
@Override  
public void deletePost(int id) {  
    List<Post> posts = getAllPosts();  
    posts.removeIf(post -> post.getId() == id);  
    saveAllPosts(posts);  
}  
  
private void saveAllPosts(List<Post> posts) {  
    try (BufferedWriter writer = new BufferedWriter(new FileWriter(FILE_PATH))) {  
        for (Post post : posts) {  
            writer.write(post.getId() + "|" + post.getTitle() + "|" + post.getDescription());  
            writer.newLine();  
        }  
    } catch (IOException e) {  
        System.err.println("Error saving posts: " + e.getMessage());  
    }  
}
```

4. Service Layer - PostService.java

Code:

```
package service;  
  
import model.Post;  
import repository.FilePostRepository;  
import repository.PostRepository;
```

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူဗာ တတ်ရမည်။



```
import java.util.List;

public class PostService {
    private PostRepository postRepository;

    public PostService() {
        this.postRepository = new FilePostRepository();
    }

    public List<Post> getAllPosts() {
        return postRepository.getAllPosts();
    }

    public Post getPostById(int id) {
        return postRepository.getPostById(id);
    }

    public void createPost(String title, String description) {
        int newId = generateNewId();
        Post post = new Post(newId, title, description);
        postRepository.addPost(post);
    }

    public void updatePost(int id, String title, String description) {
        Post post = new Post(id, title, description);
        postRepository.updatePost(post);
    }

    public void deletePost(int id) {
        postRepository.deletePost(id);
    }

    private int generateNewId() {
        List<Post> posts = getAllPosts();
        if (posts.isEmpty()) {
            return 1;
        }
        return posts.get(posts.size() - 1).getId() + 1;
    }
}
```



5. Main Application - MainApp.java

Code:

```
import model.Post;
import service.PostService;
import java.util.List;
import java.util.Scanner;

public class MainApp {
    private static PostService postService = new PostService();
    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        boolean running = true;

        while (running) {
            System.out.println("\nPost Management System");
            System.out.println("1. Create Post");
            System.out.println("2. Read All Posts");
            System.out.println("3. Read Post by ID");
            System.out.println("4. Update Post");
            System.out.println("5. Delete Post");
            System.out.println("6. Exit");
            System.out.print("Choose an option: ");

            int choice = scanner.nextInt();
            scanner.nextLine(); // consume newline

            switch (choice) {
                case 1:
                    createPost();
                    break;
                case 2:
                    readAllPosts();
                    break;
                case 3:
                    readPostById();
                    break;
            }
        }
    }

    private void createPost() {
        Post post = new Post();
        System.out.print("Enter Post Title: ");
        post.setTitle(scanner.nextLine());
        System.out.print("Enter Post Content: ");
        post.setContent(scanner.nextLine());
        postService.createPost(post);
        System.out.println("Post created successfully!");
    }

    private void readAllPosts() {
        List<Post> posts = postService.readAllPosts();
        if (posts.isEmpty()) {
            System.out.println("No posts found.");
        } else {
            for (Post post : posts) {
                System.out.println(post.getId() + ". " + post.getTitle() + " - " + post.getContent());
            }
        }
    }

    private void readPostById() {
        System.out.print("Enter Post ID: ");
        int id = scanner.nextInt();
        Post post = postService.readPostById(id);
        if (post != null) {
            System.out.println("Post ID " + post.getId() + ": " + post.getTitle() + " - " + post.getContent());
        } else {
            System.out.println("Post not found.");
        }
    }

    private void updatePost() {
        System.out.print("Enter Post ID: ");
        int id = scanner.nextInt();
        Post post = postService.readPostById(id);
        if (post == null) {
            System.out.println("Post not found.");
            return;
        }

        System.out.print("Enter new Post Title: ");
        String newTitle = scanner.nextLine();
        System.out.print("Enter new Post Content: ");
        String newContent = scanner.nextLine();

        post.setTitle(newTitle);
        post.setContent(newContent);
        postService.updatePost(post);
        System.out.println("Post updated successfully!");
    }

    private void deletePost() {
        System.out.print("Enter Post ID: ");
        int id = scanner.nextInt();
        Post post = postService.readPostById(id);
        if (post == null) {
            System.out.println("Post not found.");
            return;
        }

        postService.deletePost(post);
        System.out.println("Post deleted successfully!");
    }
}
```



```
case 4:  
    updatePost();  
    break;  
case 5:  
    deletePost();  
    break;  
case 6:  
    running = false;  
    break;  
default:  
    System.out.println("Invalid option. Please try again.");  
}  
}  
  
System.out.println("Exiting Post Management System. Goodbye!");  
}  
  
private static void createPost() {  
    System.out.print("Enter post title: ");  
    String title = scanner.nextLine();  
    System.out.print("Enter post description: ");  
    String description = scanner.nextLine();  
  
    postService.createPost(title, description);  
    System.out.println("Post created successfully!");  
}  
  
private static void readAllPosts() {  
    List<Post> posts = postService.getAllPosts();  
    if (posts.isEmpty()) {  
        System.out.println("No posts available.");  
    } else {  
        System.out.println("\nAll Posts:");  
        for (Post post : posts) {  
            System.out.println(post);  
        }  
    }  
}  
  
private static void readPostById() {
```



```
System.out.print("Enter post ID: ");
int id = scanner.nextInt();
scanner.nextLine(); // consume newline

Post post = postService.getPostById(id);
if (post != null) {
    System.out.println("\nPost Details:");
    System.out.println(post);
} else {
    System.out.println("Post not found with ID: " + id);
}
}

private static void updatePost() {
    System.out.print("Enter post ID to update: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // consume newline

    Post existingPost = postService.getPostById(id);
    if (existingPost == null) {
        System.out.println("Post not found with ID: " + id);
        return;
    }

    System.out.print("Enter new title (current: " + existingPost.getTitle() + "): ");
    String title = scanner.nextLine();
    System.out.print("Enter new description (current: " + existingPost.getDescription() + "): ");
    String description = scanner.nextLine();

    postService.updatePost(id, title, description);
    System.out.println("Post updated successfully!");
}

private static void deletePost() {
    System.out.print("Enter post ID to delete: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // consume newline

    Post existingPost = postService.getPostById(id);
    if (existingPost == null) {
```



```
        System.out.println("Post not found with ID: " + id);
        return;
    }

    postService.deletePost(id);
    System.out.println("Post deleted successfully!");
}
}
```

□ Testing Results

⇒ Test Case 1: Create Post

```
Post Management System
1. Create Post
2. Read All Posts
3. Read Post by ID
4. Update Post
5. Delete Post
6. Exit
Choose an option: 1
Enter post title: First Post
Enter post description: This is my first post
Post created successfully!
```

⇒ Test Case 2: Read All Posts

```
Post Management System
1. Create Post
2. Read All Posts
3. Read Post by ID
4. Update Post
5. Delete Post
6. Exit
Choose an option: 2
```

All Posts:

■ မြန်မာပြည်သားတိုင်း ကွန်ပျူဗာ တတ်ရမည်။



ID: 1, Title: First Post, Description: This is my first post

⇒ Test Case 3: Update Post

Post Management System

1. Create Post
2. Read All Posts
3. Read Post by ID
4. Update Post
5. Delete Post
6. Exit

Choose an option: 4

Enter post ID to update: 1

Enter new title (current: First Post): Updated First Post

Enter new description (current: This is my first post): This is an updated post

Post updated successfully!

⇒ Test Case 4: Delete Post

Post Management System

1. Create Post
2. Read All Posts
3. Read Post by ID
4. Update Post
5. Delete Post
6. Exit

Choose an option: 5

Enter post ID to delete: 1

Post deleted successfully!

⇒ Test Case 5: Read After Deletion

Post Management System

1. Create Post
2. Read All Posts
3. Read Post by ID



4. Update Post

5. Delete Post

6. Exit

Choose an option: 2

No posts available.

▣ ရှင်းလင်းချက်

1. Model Layer (Post.java): Post data structure ကို define လုပ်ထားတယ်။
2. Repository Interface (PostRepository.java): CRUD operations တွေအတွက် interface ဖြစ်တယ်။
3. File Implementation (FilePostRepository.java): File-based storage system ကို implement လုပ်ထားတယ်။ Data တွေကို pipe () နဲ့ separate လုပ်ပြီး file ထဲမှာ သိမ်းတယ်။
4. Service Layer (PostService.java): Business logic တွေကို handle လုပ်တယ်။ ID auto-generation လုပ်တဲ့ logic ပါတယ်။
5. Main Application (MainApp.java): User interface နဲ့ user input တွေကို handle လုပ်တယ်။

Data တွေကို posts_data.txt file ထဲမှာ အောက်ပါ format နဲ့ သိမ်းမယ်။

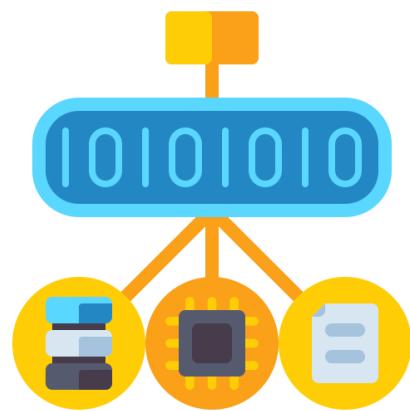
1|First Post|This is my first post

2|Second Post|Another post example

ဒါ application မှာ file-based persistence ကို အသုံးပြုထားပြီး console interface ကဲ့ CRUD operations တွေကို perform လုပ်လို့ရပါတယ်။



SERIALIZATION



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



SERIALIZATION

Java မှာ Serialization ဆိုတာက Object တစ်ခုကို Byte Stream အဖြစ် ပြောင်းလဲပြီး File ထဲသိမ်းဆည်းတဲ့ (သို့) Network ကတစ်ဆင့် ပို့လို့ရအောင် လုပ်ဆောင်ပေးတဲ့ နည်းလမ်းဖြစ်ပါတယ်။

❑ **Serialization ဘာကြောင့် အသုံးဝင်တာလဲ?**

1. Object ကို File ထဲသိမ်းလို့ရတယ်
2. Network ပေါ်ကနေ Object ပို့လို့ရတယ်
3. Database ထဲ Object အဖြစ်သိမ်းလို့ရတယ်
4. Memory ထဲ Object ကို ပြန်လည်အသုံးပြန်စိတယ်

❑ **Serialization ဘယ်လိုလုပ်မလဲ?**

1. Serializable Interface ကို Implement လုပ်ပါ

သင့် Class သဲ java.io.Serializable Interface ကို Implement လုပ်ဖို့လိုအပ်ပါတယ်။

Code:

```
import java.io.Serializable;
```

```
public class Student implements Serializable {
```

```
    private String name;
```

```
    private int age;
```

```
    public Student(String name, int age) {
```

```
        this.name = name;
```

```
        this.age = age;
```

```
}
```

```
@Override
```

```
public String toString() {
```

```
    return "Student [name=" + name + ", age=" + age + "]";
```

■ **မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။**



```
}
```

```
}
```

- Serializable သဲ Marker Interface ဖြစ်တောက်ခွင့် Method တော့ Override လုပ်စရာမလိုပါ။

2. Object ကို File အဖြစ် သိမ်းမယ် (Serialization)

- ✓ ObjectOutputStream ကို သုံးပြီး Object ကို File ထဲ ရေးမယ်။

Code:

```
import java.io.*;
```

```
public class SerializeExample {  
    public static void main(String[] args) {  
        Student student = new Student("Mg Mg", 20);  
  
        try (ObjectOutputStream oos = new ObjectOutputStream(new  
FileOutputStream("student.dat"))) {  
            oos.writeObject(student);  
            System.out.println("Object ကို File အဖြစ် သိမ်းပြီးပါပြီ!");  
        } catch (IOException e) {  
            System.out.println("Error ဖြစ်နေပါတယ်: " + e.getMessage());  
        }  
    }  
}
```

- student.dat ဆိုတဲ့ File ထဲမှာ Object သဲ Binary Format နဲ့ သိမ်းသွားမယ်။

3. File ကနေ Object ပြန်ဖတ်မယ် (Deserialization)

- ObjectInputStream သုံးပြီး File ထဲက Object ကို ပြန်ဖတ်မယ်။

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



Code:

```
import java.io.*;  
  
public class DeserializeExample {  
    public static void main(String[] args) {  
        try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream("student.dat"))) {  
            Student loadedStudent = (Student) ois.readObject();  
            System.out.println("ဖတ်လိုက်ရတဲ့ Object: " + loadedStudent);  
        } catch (IOException | ClassNotFoundException e) {  
            System.out.println("Error ဖြစ်နေပါတယ်: " + e.getMessage());  
        }  
    }  
}
```

💻 Output:

Student [name=Mg Mg, age=20]

❑ Short Notes:

- ✓ transient Keyword → ဒီ Field ကို Serialize မလုပ်စေချင်ရင် သုံးပါ။
private transient String password; // File ထဲမသိမ်းဘူး
- ✓ serialVersionUID → Class Version Control အတွက် သုံးပါ။
private static final long serialVersionUID = 1L;
- ✓ Security Risk → Unknown Source ကနေ Deserialize မလုပ်သင့်ပါ။

❑ အချုပ်ပြောရရင်

- Serialization → Object → File (Binary)
- Deserialization → File → Object
- Serializable Interface ကို သုံးရမယ်။
- transient နဲ့ Sensitive Data တွေကို ကာကွယ်နိုင်တယ်။

■ မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



လေ့ကျင့်ရန်-

အောက်ပါအတိုင်း Java console-based Post CRUD application တစ်ခုကို Serializable interface သုံးဖြီးရေးပြုမယ်။

□ Project Structure

```
post-crud-app/
├── src/
│   ├── model/
│   │   └── Post.java
│   ├── repository/
│   │   ├── PostRepository.java
│   │   └── FilePostRepository.java
│   ├── service/
│   │   └── PostService.java
│   └── MainApp.java
└── posts_data.ser
```

1. Model Class - Post.java (Serializable)

Code:

```
package model;

import java.io.Serializable;

public class Post implements Serializable {
    private static final long serialVersionUID = 1L;

    private int id;
    private String title;
    private String description;
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူဗာ တတ်ရမည်။



```
public Post(int id, String title, String description) {  
    this.id = id;  
    this.title = title;  
    this.description = description;  
}  
  
// Getters and Setters  
public int getId() {  
    return id;  
}  
  
public void setId(int id) {  
    this.id = id;  
}  
  
public String getTitle() {  
    return title;  
}  
  
public void setTitle(String title) {  
    this.title = title;  
}  
  
public String getDescription() {  
    return description;  
}  
  
public void setDescription(String description) {  
    this.description = description;  
}  
  
@Override  
public String toString() {  
    return "ID: " + id + ", Title: " + title + ", Description: " + description;  
}  
}
```



2. Repository Interface - PostRepository.java

Code:

```
package repository;

import model.Post;
import java.util.List;

public interface PostRepository {
    List<Post> getAllPosts();
    Post getPostById(int id);
    void addPost(Post post);
    void updatePost(Post post);
    void deletePost(int id);
}
```

3. File Implementation - FilePostRepository.java (Using Object Serialization)

Code:

```
package repository;

import model.Post;
import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class FilePostRepository implements PostRepository {
    private static final String FILE_PATH = "posts_data.ser";

    @Override
    public List<Post> getAllPosts() {
        List<Post> posts = new ArrayList<>();
        File file = new File(FILE_PATH);

        if (!file.exists()) {
            return posts;
        }

        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(file));
            posts = (List<Post>) ois.readObject();
            ois.close();
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
        return posts;
    }

    @Override
    public void addPost(Post post) {
        posts.add(post);
    }

    @Override
    public void updatePost(Post post) {
        posts.set(posts.indexOf(post), post);
    }

    @Override
    public void deletePost(int id) {
        posts.remove(id);
    }

    @Override
    public Post getPostById(int id) {
        return posts.get(id);
    }

    @Override
    public List<Post> getAllPosts() {
        return posts;
    }
}
```



```
}

try (ObjectInputStream ois = new ObjectInputStream(new FileInputStream(FILE_PATH))) {
    posts = (List<Post>) ois.readObject();
} catch (IOException | ClassNotFoundException e) {
    System.err.println("Error reading from file: " + e.getMessage());
}
return posts;
}

@Override
public Post getPostById(int id) {
    return getAllPosts().stream()
        .filter(post -> post.getId() == id)
        .findFirst()
        .orElse(null);
}

@Override
public void addPost(Post post) {
    List<Post> posts = getAllPosts();
    posts.add(post);
    saveAllPosts(posts);
}

@Override
public void updatePost(Post updatedPost) {
    List<Post> posts = getAllPosts();
    for (int i = 0; i < posts.size(); i++) {
        if (posts.get(i).getId() == updatedPost.getId()) {
            posts.set(i, updatedPost);
            break;
        }
    }
    saveAllPosts(posts);
}

@Override
public void deletePost(int id) {
    List<Post> posts = getAllPosts();
```



```
posts.removeIf(post -> post.getId() == id);
saveAllPosts(posts);
}

private void saveAllPosts(List<Post> posts) {
    try (ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(FILE_PATH))) {
        oos.writeObject(posts);
    } catch (IOException e) {
        System.err.println("Error saving posts: " + e.getMessage());
    }
}
```

4. Service Layer - PostService.java

Code:

```
package service;

import model.Post;
import repository.FilePostRepository;
import repository.PostRepository;
import java.util.List;

public class PostService {
    private PostRepository postRepository;

    public PostService() {
        this.postRepository = new FilePostRepository();
    }

    public List<Post> getAllPosts() {
        return postRepository.getAllPosts();
    }

    public Post getPostById(int id) {
        return postRepository.getPostById(id);
    }
}
```



```
public void createPost(String title, String description) {  
    int newId = generateNewId();  
    Post post = new Post(newId, title, description);  
    postRepository.addPost(post);  
}  
  
public void updatePost(int id, String title, String description) {  
    Post post = new Post(id, title, description);  
    postRepository.updatePost(post);  
}  
  
public void deletePost(int id) {  
    postRepository.deletePost(id);  
}  
  
private int generateNewId() {  
    List<Post> posts = getAllPosts();  
    if (posts.isEmpty()) {  
        return 1;  
    }  
    return posts.get(posts.size() - 1).getId() + 1;  
}
```

5. Main Application - MainApp.java

Code:

```
import model.Post;  
import service.PostService;  
import java.util.List;  
import java.util.Scanner;  
  
public class MainApp {  
    private static PostService postService = new PostService();  
    private static Scanner scanner = new Scanner(System.in);  
  
    public static void main(String[] args) {  
        boolean running = true;
```



```
while (running) {  
    System.out.println("\nPost Management System");  
    System.out.println("1. Create Post");  
    System.out.println("2. Read All Posts");  
    System.out.println("3. Read Post by ID");  
    System.out.println("4. Update Post");  
    System.out.println("5. Delete Post");  
    System.out.println("6. Exit");  
    System.out.print("Choose an option: ");  
  
    int choice = scanner.nextInt();  
    scanner.nextLine(); // consume newline  
  
    switch (choice) {  
        case 1:  
            createPost();  
            break;  
        case 2:  
            readAllPosts();  
            break;  
        case 3:  
            readPostById();  
            break;  
        case 4:  
            updatePost();  
            break;  
        case 5:  
            deletePost();  
            break;  
        case 6:  
            running = false;  
            break;  
        default:  
            System.out.println("Invalid option. Please try again.");  
    }  
}  
  
System.out.println("Exiting Post Management System. Goodbye!");  
}
```



```
private static void createPost() {  
    System.out.print("Enter post title: ");  
    String title = scanner.nextLine();  
    System.out.print("Enter post description: ");  
    String description = scanner.nextLine();  
  
    postService.createPost(title, description);  
    System.out.println("Post created successfully!");  
}  
  
private static void readAllPosts() {  
    List<Post> posts = postService.getAllPosts();  
    if (posts.isEmpty()) {  
        System.out.println("No posts available.");  
    } else {  
        System.out.println("\nAll Posts:");  
        for (Post post : posts) {  
            System.out.println(post);  
        }  
    }  
}  
  
private static void readPostById() {  
    System.out.print("Enter post ID: ");  
    int id = scanner.nextInt();  
    scanner.nextLine(); // consume newline  
  
    Post post = postService.getPostById(id);  
    if (post != null) {  
        System.out.println("\nPost Details:");  
        System.out.println(post);  
    } else {  
        System.out.println("Post not found with ID: " + id);  
    }  
}  
  
private static void updatePost() {  
    System.out.print("Enter post ID to update: ");  
    int id = scanner.nextInt();  
    scanner.nextLine(); // consume newline
```



```
Post existingPost = postService.getPostById(id);
if (existingPost == null) {
    System.out.println("Post not found with ID: " + id);
    return;
}

System.out.print("Enter new title (current: " + existingPost.getTitle() + "): ");
String title = scanner.nextLine();
System.out.print("Enter new description (current: " + existingPost.getDescription() + "): ");
String description = scanner.nextLine();

postService.updatePost(id, title, description);
System.out.println("Post updated successfully!");
}

private static void deletePost() {
    System.out.print("Enter post ID to delete: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // consume newline

    Post existingPost = postService.getPostById(id);
    if (existingPost == null) {
        System.out.println("Post not found with ID: " + id);
        return;
    }

    postService.deletePost(id);
    System.out.println("Post deleted successfully!");
}
}
```



❑ Testing Results

⇒ Test Case 1: Create Posts

Post Management System

1. Create Post
2. Read All Posts
3. Read Post by ID
4. Update Post
5. Delete Post
6. Exit

Choose an option: 1

Enter post title: Java Programming

Enter post description: Learning Java basics

Post created successfully!

Choose an option: 1

Enter post title: OOP Concepts

Enter post description: Understanding Object Oriented Programming

Post created successfully!

⇒ Test Case 2: Read All Posts

Choose an option: 2

All Posts:

ID: 1, Title: Java Programming, Description: Learning Java basics

ID: 2, Title: OOP Concepts, Description: Understanding Object Oriented Programming

⇒ Test Case 3: Update Post

Choose an option: 4

Enter post ID to update: 1

Enter new title (current: Java Programming): Java Fundamentals



Enter new description (current: Learning Java basics): Basic Java programming concepts
Post updated successfully!

Choose an option: 2

All Posts:

ID: 1, Title: Java Fundamentals, Description: Basic Java programming concepts
ID: 2, Title: OOP Concepts, Description: Understanding Object Oriented Programming

⇒ Test Case 4: Delete Post

Choose an option: 5

Enter post ID to delete: 2

Post deleted successfully!

Choose an option: 2

All Posts:

ID: 1, Title: Java Fundamentals, Description: Basic Java programming concepts

⇒ Test Case 5: Restart Application and Verify Persistence

အပေါ်က test တွေ ပြလုပ်ပြီးတဲ့နောက် application ကို restart လုပ်ပြီး data တွေ persist ဖြစ်နေတာကို စစ်ဆေးနိုင်ပါတယ်။

Post Management System

1. Create Post
2. Read All Posts
3. Read Post by ID
4. Update Post
5. Delete Post
6. Exit

Choose an option: 2



All Posts:

ID: 1, Title: Java Fundamentals, Description: Basic Java programming concepts

▣ ရှင်းလင်းချက်

1. Serializable Implementation: Post class ကို Serializable interface သုံးပြီး implement လုပ်ထားတယ်။ serialVersionUID ကို define လုပ်ထားတယ်။
2. File Storage: Data တွေကို binary format နဲ့ posts_data.ser file ထဲမှာ Object Serialization သုံးပြီး သိမ်းတယ်။
3. CRUD Operations:
 - o Create: ObjectOutputStream သုံးပြီး list of posts တစ်ခုလုံးကို file ထဲမှာ သိမ်းတယ်။
 - o Read: ObjectInputStream သုံးပြီး file ထဲက data တွေကို ဖတ်တယ်။
 - o Update: Post object ကို update လုပ်ပြီး list တစ်ခုလုံးကို ပြန်သိမ်းတယ်။
 - o Delete: Post ကို list ထဲက ဖျက်ပြီး list တစ်ခုလုံးကို ပြန်သိမ်းတယ်။
4. Advantages of Serialization:
 - o Complex object တွေကို အလွယ်တကူ သိမ်းလိုက်တယ်။
 - o Data structure တွေကို မပျက်မစီး သိမ်းဆည်းနိုင်တယ်။
 - o Text file သုံးတာထက် ပိုပြီး efficient ဖြစ်တယ်။

ဒါ application မှာ object serialization ကို အသုံးပြုထားတဲ့ အတွက် data တွေကို binary format နဲ့ သိမ်းဆည်းပြီး application restart လုပ်တဲ့ အခါမှုလည်း မူလ data တွေကို ပြန်ရနိုင်ပါတယ်။



EXCEPTION HANDLING



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



EXCEPTION HANDLING

Exception Handling ဆိတာက program လုပ်ဆောင်နေစဉ် ဖြစ်ပေါ်လာနိုင်တဲ့ အမှားတွေ (errors) ကို
ထိန်းချုပ်ဖြေရှင်းနည်းပါ။ Exception တွေကို ကောင်းစွာ handle လုပ်ခြင်းဖြင့် program crash မဖြစ်အောင်
ကာကွယ်နိုင်ပါတယ်။

□ Exception အမျိုးအစားများ

1. Checked Exceptions (Compile Time Exceptions)

- Compile time မှာ စစ်ဆေးတဲ့ exceptions
- ဥပမာ: IOException, SQLException, ClassNotFoundException

2. Unchecked Exceptions (Runtime Exceptions)

- Runtime မှာ ဖြစ်ပေါ်တဲ့ exceptions
- ဥပမာ: NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException

3. Errors

- System level errors (ပြင်ဆင်လို့မရတဲ့ အမှားများ)
- ဥပမာ: OutOfMemoryError, StackOverflowError

Exception Handling အဓိက Keywords

Keyword	ရှင်းလင်းချက်
try	Exception ဖြစ်နိုင်တဲ့ code block
catch	Exception ကို handle လုပ်မယ့် block
finally	Exception ဖြစ်ဖြစ်/မဖြစ်ဖြစ် အမြဲ run မယ့် block
throw	Manual exception throw လုပ်ဖို့

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



Keyword	ရှင်းလင်းချက်
throws	Method က exception throw လုပ်နိုင်ကြောင်း declare လုပ်ဖို့

□ Exception Handling ဥပမာဏ်း

1. အခြေခံ try-catch အသုံးပြုပုံ

Code:

```
try {  
    int result = 10 / 0; // ArithmeticException ဖြစ်မယ်  
} catch (ArithmaticException e) {  
    System.out.println("သူညနဲ့တော်လို့မရဘူး: " + e.getMessage());  
}
```

2. Multiple catch blocks

Code:

```
try {  
    int[] arr = new int[5];  
    arr[10] = 50; // ArrayIndexOutOfBoundsException  
} catch (ArrayIndexOutOfBoundsException e) {  
    System.out.println("Array index မှားနေပါတယ်");  
} catch (Exception e) {  
    System.out.println("တွေး exception တစ်ခုဗုံဖြစ်နေပါတယ်");  
}
```

3. finally block အသုံးပြုပုံ

Code:

```
try {  
    // File operations  
} catch (IOException e) {  
    System.out.println("File error: " + e);  
} finally {
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



```
System.out.println("ဒီ code အမြဲ run မယ်");
}
```

4. throw နဲ့ custom exception

Code:

```
void checkAge(int age) {
    if (age < 18) {
        throw new ArithmeticException("အသက်မပြည့်သေးပါ");
    } else {
        System.out.println("ဝင်ရောက်ခွင့်ပြုပါသည်");
    }
}

public static void main(String[] args) {
    checkAge(15); // Exception throw လုပ်မယ်
}
```

5. throws keyword အသုံးပြုပဲ

Code:

```
void readFile() throws IOException {
    File file = new File("nonexistent.txt");
    FileReader fr = new FileReader(file); // IOException ဖြစ်တယ်
}
```

Custom Exception ဖန်တီးနည်း

Code:

```
class MyCustomException extends Exception {
    public MyCustomException(String message) {
        super(message);
    }
}
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးဘာ တတ်ရမည်။



```
// အသံပြုပုံ  
try {  
    throw new MyCustomException("ကျန်တော့် custom exception");  
} catch (MyCustomException e) {  
    System.out.println(e.getMessage());  
}
```

☞ Short Notes:

1. Specific exceptions တွေကို ဦးစားပေးဖမ်းပါ
2. Resource တွေကို finally block မှာ close လုပ်ပါ
3. Exception messages တွေကို ရှင်းလင်းစွာရေးပါ
4. အလွန်အကျိုး try-catch မလုပ်ပါနဲ့
5. Logging ကိုအသံပြုပါ

Java Exception Handling ကို ကောင်းစွာနားလည်ထားရင် သင့် program တွေကို ပိုမိုခိုင်မာစွာ ရေးသားနိုင်မယ်ဖြစ်ပါတယ!



TESTING & DEBUGGING



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



TESTING AND DEBUGGING

1. Unit Testing (JUnit 5)

Java မှာ Unit Test ရေးစွဲ JUnit 5 ကို အသုံးပြုပါတယ်။

⇒ အခြေခံ Test Case ရေးနည်း

Code:

```
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*;  
  
class CalculatorTest {  
  
    @Test  
    void testAddition() {  
        Calculator calc = new Calculator();  
        assertEquals(5, calc.add(2, 3)); // 2+3 = 5 ဖြစ်ရမယ်  
    }  
  
    @Test  
    void testDivision() {  
        Calculator calc = new Calculator();  
        assertThrows(ArithmeticException.class, () -> calc.divide(10, 0)); // 0-ခဲ့စားရင် Exception ပြန်ရမယ်  
    }  
}
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးဘာ တတ်ရမည်။



❑ JUnit 5 အခိုက် Annotations

✓ Annotation အသုံးဝင်ပုံ

Annotation	အသုံးဝင်ပုံ
------------	-------------

@Test	Test Method အဖြစ်သတ်မှတ်
-------	--------------------------

@BeforeEach	Test တစ်ခုမစစေ Run
-------------	--------------------

@AfterEach	Test တစ်ခုပြီးတိုင်း Run
------------	--------------------------

@Disabled	Test ကို ယခုအချိန်မစစ်
-----------	------------------------

2. Integration Testing

Spring Boot သုံးရင် @SpringBootTest နဲ့ Test လုပ်နည်ပါတယ်။

Code:

```
@SpringBootTest
class UserServiceIntegrationTest {

    @Autowired
    private UserService userService;

    @Test
    void testUserCreation() {
        User user = userService.createUser("John", "john@example.com");
        assertNotNull(user.getId());
    }
}
```

3. Debugging Techniques

Breakpoints သုံးနည်း (Eclipse/IntelliJ)

■ မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



Line Breakpoint - လိုချင်တဲ့ Line မှာ Click ချက်

Conditional Breakpoint - Condition သတ်မှတ်ပြီး Break

Exception Breakpoint - Exception ဖြစ်တိုင်း Break

⇒ Logging သံနည်း (SLF4J)

Code:

```
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
public class OrderService {  
    private static final Logger logger = LoggerFactory.getLogger(OrderService.class);  
  
    public void processOrder(Order order) {  
        logger.debug("Processing order: {}", order.getId());  
        try {  
            // business logic  
            logger.info("Order processed successfully");  
        } catch (Exception e) {  
            logger.error("Error processing order", e);  
        }  
    }  
}
```

4. Assertion Methods အမျိုးမျိုး

Method	ရည်ရွယ်ချက်
assertEquals(expected, actual)	တူရမယ်
assertTrue(condition)	True ဖြစ်ရမယ်
assertNull(object)	Null ဖြစ်ရမယ်
assertThrows(Exception.class, executable)	Exception ဖြန်ရမယ်



5. Mocking (Mockito)

Test Double ဆွေအတွက် Mockito သုံးပါ။

Code:

```
@ExtendWith(MockitoExtension.class)
class PaymentServiceTest {

    @Mock
    private PaymentGateway paymentGateway;

    @InjectMocks
    private PaymentService paymentService;

    @Test
    void testSuccessfulPayment() {
        when(paymentGateway.process(anyDouble())).thenReturn(true);
        assertTrue(paymentService.makePayment(100.0));
    }
}
```

6. Test Coverage

JaCoCo ဆုံးပြုး Test Coverage စုစုပေါင်း။

```
xml
<!-- pom.xml -->
<plugin>
    <groupId>org.jacoco</groupId>
    <artifactId>jacoco-maven-plugin</artifactId>
    <version>0.8.7</version>
    <executions>
        <execution>
            <goals>
```



```
<goal>prepare-agent</goal>
</goals>
</execution>
<execution>
<id>report</id>
<phase>test</phase>
<goals>
<goal>report</goal>
</goals>
</execution>
</executions>
</plugin>
```

7. Debugging Tips

Stack Trace ကို သေချာဖတ်ပါ

Step Over (F6), Step Into (F5) ကိုသုံးပါ

Variables View ကနေ Value တွေစစ်ပါ

Evaluate Expression သုံးပြီး Code စမ်းပါ

▣ အကုန်ခြိုးပြောရရင်

- JUnit 5 နဲ့ Unit Test ရေးပါ
- Integration Test အတွက် @SpringBootTest သုံးပါ
- Mockito နဲ့ Dependency တွေကို Mock လုပ်ပါ
- Logging နဲ့ Debugging လုပ်ပါ
- JaCoCo သုံးပြီး Coverage စစ်ပါ



THREAD



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



THREAD

Java မှာ Thread ဆိတာက program execution ၏ အသေးယ်ဆုံး unit တစ်ခုဖြစ်ပြီး multitasking လုပ်ဆောင်နိုင်ဖို့အတွက် အသုံးပြုပါတယ်။ Thread တွေကို အသုံးပြုခြင်းဖြင့် ကွွန်တော် တို့ program တစ်ခုတည်းမှာ တစ်ချိန်တည်း task များစွာကို လုပ်ဆောင်နိုင်ပါတယ်။

□ Thread အမျိုးအစားများ

1. User Thread (Non-Daemon Thread)

- Main program ရဲ့အလုပ်တွေကို လုပ်ဆောင်တဲ့ thread
- JVM ၏ thread အားလုံး ပြီးဆုံးမှသာ program ၏ ရပ်တန်မယ်

2. Daemon Thread

- Background service တွေအတွက် အသုံးပြုတဲ့ thread
- User thread တွေ အားလုံးပြီးသွားရင် အလိုအလျောက် ရပ်တန်သွားမယ်

□ Thread ဖန်တီးနည်းများ

1. Thread Class ကို Extend လုပ်ခြင်း

Code:

```
class MyThread extends Thread {  
    public void run() {  
        System.out.println("Thread is running...");  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        MyThread t1 = new MyThread();  
        t1.start(); // Thread စပါမယ်  
    }  
}
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



2. Runnable Interface ကို Implement လုပ်ခြင်း (ပိုကောင်းတဲ့နည်း)

Code:

```
class MyRunnable implements Runnable {  
    public void run() {  
        System.out.println("Runnable thread is running...");  
    }  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Thread t2 = new Thread(new MyRunnable());  
        t2.start();  
    }  
}
```

Thread အစိတ် Method များ

Method	ရှင်းလင်းချက်
start()	Thread ကို စတင်မယ်
run()	Thread ရဲ့ အလုပ်တွေကို သတ်မှတ်မယ်
sleep(long millis)	Thread ကို သတ်မှတ်ထားတဲ့ အချိန်ကြာမှ ပြန်စမယ်
join()	အခြား thread ပြီးတဲ့ အထိ စောင့်မယ်
interrupt()	Thread ကို အန္တံ့အယူက်ပေးမယ်
isAlive()	Thread အလုပ်လုပ်နေခဲ့လားစစ်မယ်

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



❑ Thread Synchronization (အချိန်ကိုက်ညီခြင်း)

Multiple threads တွေ shared resource တစ်ခုကို တစ်ချိန်တည်း access လုပ်တဲ့အခါ race condition ဖြစ်နိုင်ပါတယ်။ ဒါကို ကာကွယ်ဖို့ synchronization လုပ်ရပါမယ်။

❑ synchronized Method အသုံးပြုပုံ

Code:

```
class Counter {  
    private int count = 0;  
  
    public synchronized void increment() {  
        count++;  
    }  
  
    public int getCount() {  
        return count;  
    }  
}
```

❑ synchronized Block အသုံးပြုပုံ

Code:

```
public void doWork() {  
    synchronized(this) {  
        // synchronized code block  
    }  
}
```

Thread Life Cycle (သက်တမ်းစက်ဝန်း)

1. New - Thread object ဖုန်တီးပြီး start() မခေါ်ရသေးခြင်း
2. Runnable - start() ခေါ်ပြီး CPU ရဖို့တောင့်နေခြင်း
3. Running - CPU ရပြီး execute လုပ်နေခြင်း

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



4. Blocked/Waiting - I/O operation သို့မဟုတ် synchronization ကြောင့်စောင့်နေခြင်း
5. Terminated - run() method ပြီးဆုံးခြင်း

□ Thread Pool အကြောင်း

Thread တွေကို ထပ်ခါထပ်ခါ create/destroy လုပ်တာက performance ကိုထိခိုက်စေနိုင်ပါတယ်။ Thread pool ကို အသုံးပြုခြင်းဖြင့် ဒီပြဿနာကို ဖြေရှင်းနိုင်ပါတယ်။

Code:

```
ExecutorService executor = Executors.newFixedThreadPool(5);

for (int i = 0; i < 10; i++) {
    Runnable worker = new WorkerThread("" + i);
    executor.execute(worker);
}
executor.shutdown();
```

□ ဘယ်အချင့်မှာ Thread ကိုသုံးမလဲ?

- ✓ GUI applications မှာ responsiveness မြင့်စိုး
- ✓ Server applications မှာ multiple clients ကို handle လုပ်စိုး
- ✓ ကြာမြင့်တဲ့ operations တွေကို background မှာ run စိုး
- ✓ Parallel processing လုပ်စိုး

Java Thread ကို ကောင်းစွာနားလည်ထားရင် သင့် program တွေကို ပိုမိုထိရောက်စွာ ရေးသားနိုင်မယ်ဖြစ်ပါတယ်!



လေ့ကျင့်ရန်

အောက်ပါ program မှာ Java Thread အသုံးပြုပုံကို ရှင်းပြထားပါတယ်။ Thread နှစ်ခုဖြင့် parallel processing လုပ်ဆောင်ပုံကို ဥပမာပြထားပါတယ်။

Code:

```
class ThreadDemo {  
    public static void main(String[] args) {  
        // Thread နှစ်ခု ဖွံ့ဖြိုးမယ်  
        Thread thread1 = new NumberPrinter("Thread-1");  
        Thread thread2 = new NumberPrinter("Thread-2");  
  
        // Thread စွဲတို့ စေမယ်  
        thread1.start();  
        thread2.start();  
  
        System.out.println("Main thread မှ စတင်ပြီးပါပြီ...");  
    }  
}  
  
class NumberPrinter extends Thread {  
    private String threadName;  
  
    public NumberPrinter(String name) {  
        this.threadName = name;  
        System.out.println(threadName + " ဖွံ့ဖြိုးပါပြီ");  
    }  
  
    @Override
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



```
public void run() {  
    System.out.println(threadName + " စတင်လုပ်ဆောင်နေပါတယ်");  
  
    try {  
        for(int i = 1; i <= 5; i++) {  
            System.out.println(threadName + ": " + i);  
            // Thread ကို ခဲ့ရပ်ထားမယ်  
            Thread.sleep(500);  
        }  
    } catch (InterruptedException e) {  
        System.out.println(threadName + " အား အနောင့်အယျက်ဖြစ်ခဲ့တယ်");  
    }  
  
    System.out.println(threadName + " လုပ်ဆောင်မှု ပြီးဆုံးပါပြီ");  
}  
}
```

□ အသေးစိတ်ရှင်းလင်းချက်

1. Thread Class:

- NumberPrinter class သည် Thread class ကို extend လုပ်ထားပါတယ်။
- run() method ကို override လုပ်ပြီး thread ခဲ့အလုပ်တွေကို သတ်မှတ်ထားပါတယ်။

2. Thread Creation:

- Thread thread1 = new NumberPrinter("Thread-1"); - Thread အသစ်တစ်ခု ဖန်တီးပါတယ်။
- thread1.start(); - Thread ကို စတင်လုပ်ဆောင်စေပါတယ်။

3. Thread Execution:

- Thread တစ်ခုစီမှာ 1 မှ 5 အထိ နံပါတ်တွေကို print ထုတ်ပါတယ်။
- Thread.sleep(500); - 500 milliseconds (0.5 second) စောင့်ပါတယ်။

4. Output Example:

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
Thread-1 ဖန်တီးပြီးပါပြီ  
Thread-2 ဖန်တီးပြီးပါပြီ  
Main thread မှ စတင်ပြီးပါပြီ...  
Thread-1 စတင်လုပ်ဆောင်နေပါတယ်  
Thread-2 စတင်လုပ်ဆောင်နေပါတယ်  
Thread-1: 1  
Thread-2: 1  
Thread-1: 2  
Thread-2: 2  
Thread-1: 3  
Thread-2: 3  
Thread-1: 4  
Thread-2: 4  
Thread-1: 5  
Thread-2: 5  
Thread-1 လုပ်ဆောင်မှု ပြီးဆုံးပါပြီ  
Thread-2 လုပ်ဆောင်မှု ပြီးဆုံးပါပြီ
```

Thread နှစ်ခုဟာ တစ်ပြိုင်နက် run နေတာကို မြင်ရမှာဖြစ်ပါတယ်။

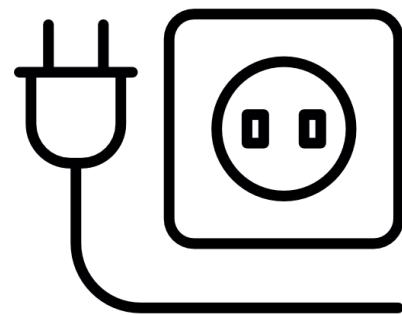
5. Thread.sleep():

- Thread ကို ခဏရပ်ထားဖို့ အသုံးပြုပါတယ်။
- InterruptedException ကို handle လုပ်ဖို့လိုပါတယ်။

ဤ program ကို run လိုက်ပါက thread နှစ်ခုဟာ တစ်ပြိုင်နက် အလုပ်လုပ်နေတာကို မြင်ရမှာဖြစ်ပါတယ်။ Thread တစ်ခုစီမှာ ကိုယ်ပိုင် execution flow ရှိပြီး CPU resources တွေကို ဝေမျှသုံးစွဲပါတယ်။



SOCKET



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



SOCKET

Java Socket Programming သဲ network communication အတွက် အသုံးဝင်တဲ့ concept တစ်ခုဖြစ်ပါတယ်။ client နဲ့ server တို့ကြား data တွေကို TCP/IP or UDP protocol တွေသုံးပြီး ပို့ဆောင်ရှိအတွက် Java မှာ java.net package ကိုအသုံးပြုပါတယ်။

- Socket နဲ့ ServerSocket အလုပ်လုပ်ပုံ

Java Socket Programming မှာ ServerSocket သဲ server side မှာ client တွေရဲ့ connection request တွေကို လက်ခံဖို့အတွက်သုံးပါတယ်။ Socket ကတော့ client side ကနေ server နဲ့ connect လုပ်ဖို့အတွက်သုံးပါတယ်။

⇒ Server-Side Code Example

Code:

```
import java.io.*;
import java.net.*;

public class Server {
    public static void main(String[] args) throws IOException {
        ServerSocket serverSocket = new ServerSocket(1234); // Port 1234 ကိုဖွင့်ထားတယ်
        System.out.println("Server is waiting for client...");

        Socket clientSocket = serverSocket.accept(); // Client connection ကိုစောင့်နေတယ်
        System.out.println("Client connected!");

        // Data ပွဲဖို့ PrintWriter သုံးတယ်
        PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
        out.println("Hello from Server!");

        // Data ဖတ်ဖို့ BufferedReader သုံးတယ်
        BufferedReader in = new BufferedReader(new
        InputStreamReader(clientSocket.getInputStream()));
        String clientMessage = in.readLine();
        System.out.println("Client says: " + clientMessage);
    }
}
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
// Resources တွက်ပိတ်တယ်  
in.close();  
out.close();  
clientSocket.close();  
serverSocket.close();  
}  
}
```

⇒ Client-Side Code Example

Code:

```
import java.io.*;  
import java.net.*;  
  
public class Client {  
    public static void main(String[] args) throws IOException {  
        Socket socket = new Socket("localhost", 1234); // Server ရဲ IP နဲ့ port ကိုသုံးပြီး connect လုပ်တယ်  
  
        // Server ကပိုလိုက်တဲ့ data ကိုဖတ်တယ်  
        BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
        String serverMessage = in.readLine();  
        System.out.println("Server says: " + serverMessage);  
  
        // Server ကို response ပြန်ပို့တယ်  
        PrintWriter out = new PrintWriter(socket.getOutputStream(), true);  
        out.println("Hello from Client!");  
  
        // Resources တွက်ပိတ်တယ်  
        in.close();  
        out.close();  
        socket.close();  
    }  
}
```

⇒ TCP vs UDP Sockets

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



Java မှာ TCP (Stream Sockets) နဲ့ UDP (Datagram Sockets) ဆိုပြီး နှစ်မျိုးရှိပါတယ်။

- TCP (ServerSocket & Socket) → Reliable, connection-oriented (ဥပမာ: Web browsing, file transfer)
- UDP (DatagramSocket & DatagramPacket) → Fast, connectionless (ဥပမာ: Video streaming, online gaming)

□ အရေးကြီးတဲ့ Java Socket Classes

- Socket → Client-side connection
- ServerSocket → Server-side connection listener
- DatagramSocket → UDP communication
- DatagramPacket → UDP data packets ပို/လက်ခံစိုး

□ Short Notes

- Port Numbers (0-65535) → Well-known ports (0-1023) ကိုရောင်ပြီး high-numbered ports သုံးသင့်တယ်။
- Exception Handling → IOException ကို အမြဲ handle လုပ်ပါ။
- Multi-Threading → Multiple clients ကို handle လုပ်ဖို့ threads သုံးနိုင်တယ်။

Java Socket Programming ကို လေ့လာပြီး chat applications, file transfer systems, နဲ့ multiplayer games တွေအတွက် အသုံးချိန်ပါတယ်။ လက်တွေ့လေ့ကျင့်ဖို့ simple client-server project တစ်ခုစလုပ်ကြည့်ပါ!



လေ့ကျင့်ရန်

Java Socket Application (Client-Server)

အောက်ပါ program မှာ Java Socket အသုံးပြုပြီး Client-Server Communication လုပ်ဆောင်ပုံကို ရှင်းပြထားပါတယ်။

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



⇒ Server Code

```
import java.io.*;
import java.net.*;

public class SimpleServer {
    public static void main(String[] args) {
        try {
            // ServerSocket ဖနတီးမယ် (port 12345)
            ServerSocket serverSocket = new ServerSocket(12345);
            System.out.println("Server စတင်ပြီးပါမြို့... (Port 12345 ကို စောင့်နေပါတယ်)");

            // Client connection ရှိ စောင့်မယ်
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client ချိတ်ဆက်လာပါမြို့: " + clientSocket.getInetAddress());

            // Input/Output streams ဖနတီးမယ်
            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

            // Client ဆိုတဲ့ message ရှိ ဖတ်မယ်
            String clientMessage = in.readLine();
            System.out.println("Client ဆိုကရတဲ့ message: " + clientMessage);

            // Client ဆိုတဲ့ response ပြန်စိုးမယ်
            out.println("Server ဆိုကနေ ပြန်ပြောတပါ: " + clientMessage.toUpperCase());

            // Connections ထွေ ပိတ်မယ်
            in.close();
            out.close();
            clientSocket.close();
            serverSocket.close();
        } catch (IOException e) {
            System.out.println("Server error: " + e.getMessage());
        }
    }
}
```



```
    }  
}
```

⇒ Client Code

```
import java.io.*;  
import java.net.*;  
  
public class SimpleClient {  
    public static void main(String[] args) {  
        try {  
            // Server ဆီကို connect လုပ်မယ် (localhost, port 12345)  
            Socket socket = new Socket("localhost", 12345);  
            System.out.println("Server ဆီကို ချိတ်ဆက်ပြီးပါပြီ");  
  
            // Input/Output streams ဖန်တီးမယ်  
            BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
;  
            PrintWriter out = new PrintWriter(socket.getOutputStream(), true);  
  
            // Server ဆီရှိ message ရှိမယ်  
            String messageToSend = "Hello Server!";  
            out.println(messageToSend);  
            System.out.println("Server ဆီကို ပိုလိုက်တဲ့ message: " + messageToSend);  
  
            // Server ဆီ၏ response ကို ဖတ်မယ်  
            String serverResponse = in.readLine();  
            System.out.println("Server ဆီကရတဲ့ response: " + serverResponse);  
  
            // Connections ထွေ ခိတ်မယ်  
            in.close();  
            out.close();  
            socket.close();  
        } catch (IOException e) {  
            System.out.println("Client error: " + e.getMessage());  
        }  
    }  
}
```

■ မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



}

▣ အသေးစိတ်ရှင်းလင်းချက်

1. Server Side:

- ServerSocket ကို အသုံးပြု၍ specific port (12345) မှာ စောင့်နေပါတယ်
- accept() method ၏ client connection ကို စောင့်နေပါတယ်
- Client ဆီက message ကို BufferedReader နဲ့ ဖတ်ပါတယ်
- Response ကို PrintWriter နဲ့ ပြန်ပို့ပါတယ်

2. Client Side:

- Socket class ကို အသုံးပြု၍ server ဆီကို connect လုပ်ပါတယ်
- Message ပိုမို PrintWriter ကို အသုံးပြုပါတယ်
- Server ဆီက response ကို BufferedReader နဲ့ ဖတ်ပါတယ်

3. အလုပ်လုပ်ပုံ အဆင့်ဆင့်:

1. Server ကို အရင် run ပါ (အခြား terminal မှာ)
2. Client ကို run ပါ
3. Client ၏ message ပိုမိုယူ
4. Server ၏ message ကို လက်ခံပြီး response ပြန်ပို့မယ်
5. Client ၏ response ကို လက်ခံမယ်

4. Output Example:

⇒ Server Output:

Server စတင်ပြီးပါပြီ... (Port 12345 ကို စောင့်နေပါတယ်)

Client ချိတ်ဆက်လာပါပြီ: /127.0.0.1

Client ဆီကရတဲ့ message: Hello Server!

⇒ Client Output:

Server ဆီကို ချိတ်ဆက်ပြီးပါပြီ

Server ဆီကို ပိုလိုက်တဲ့ message: Hello Server!

Server ဆီကရတဲ့ response: Server ဆီကနေ ပြန်ပြောတာပါ: HELLO SERVER!



JDBC



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။

 JDBC

JDBC ဆိုတာ Java Application တွေကနေ Database တွေနဲ့ချိတ်ဆက်အလုပ်လုပ်နည်အောင် Java ကပေးတဲ့ Standard API ဖြစ်ပါတယ်။

□ JDBC ၏ အခြေခံအစိတ်အပိုင်းများ

1. DriverManager:

- Database connection တွေကိုစီမံပေးပါတယ်
- Class.forName("com.mysql.jdbc.Driver"); လိုသံဃှေးပြီး driver load လုပ်ရပါတယ်

2. Connection:

- Database နဲ့ချိတ်ဆက်မှုကိုကိုယ်စားပြုပါတယ်
- Connection conn = DriverManager.getConnection(url, username, password);

3. Statement:

- SQL query တွေကို execute လုပ်ဖို့သံဃှေးပါတယ်
- Statement stmt = conn.createStatement();

4. ResultSet:

- Database ကပြန်လာတဲ့ result data တွေကိုလက်ခံပါတယ်
- ResultSet rs = stmt.executeQuery("SELECT * FROM users");



❑ JDBC အသုံးပြုပြုချေမှု (MySQL)

```
import java.sql.*;  
  
public class JdbcExample {  
    public static void main(String[] args) {  
        String url = "jdbc:mysql://localhost:3306/mydatabase";  
        String username = "root";  
        String password = "password";  
  
        try {  
            // 1. Driver load လုပ်ခြင်း  
            Class.forName("com.mysql.cj.jdbc.Driver");  
  
            // 2. Database နဲ့ဆိတ်ဆက်ခြင်း  
            Connection conn = DriverManager.getConnection(url, username, password);  
  
            // 3. Statement ဖန်တီးခြင်း  
            Statement stmt = conn.createStatement();  
  
            // 4. SQL Query မှု execute လုပ်ခြင်း  
            ResultSet rs = stmt.executeQuery("SELECT id, name, email FROM users");  
  
            // 5. Result စုစုပေါင်ခြင်း  
            while(rs.next()) {  
                int id = rs.getInt("id");  
                String name = rs.getString("name");  
                String email = rs.getString("email");  
                System.out.println("ID: " + id + ", Name: " + name + ", Email: " + email);  
            }  
  
            // 6. Resources များကိုပါတ်ခြင်း  
            rs.close();  
            stmt.close();  
            conn.close();  
  
        } catch (ClassNotFoundException e) {  
            System.out.println("MySQL Driver not found!");  
        }  
    }  
}
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
        e.printStackTrace();
    } catch (SQLException e) {
        System.out.println("Database connection failed!");
        e.printStackTrace();
    }
}
```

❑ JDBC ၏ အားသာချက်များ

1. Standard API:

- Database အမျိုးမျိုးနှင့်အလုပ်လုပ်နိုင်ပါတယ် (MySQL, Oracle, PostgreSQL စသည်)

2. Performance:

- Direct database connection ကြောင့် performance ကောင်းပါတယ်

3. Full Control:

- SQL query တွေကိုတိုက်ရှိက်ရေးနိုင်ပါတယ်

❑ JDBC ၏ အားနည်းချက်များ

1. Boilerplate Code:

- Connection, Statement, ResultSet တွေကို manual ဝိတ်ပေးရပါတယ်
- Try-with-resources သုံးရင်ပါကောင်းပါတယ်

2. No Object Mapping:

- ResultSet ကနေ Java object အဖြစ်အလိုအလျောက်မပြောင်းနိုင်ပါ

❑ JDBC အဆင့်မြင့်အသုံးပြုနည်းများ

1. PreparedStatement:

- SQL injection ကိုကာကွယ်ဖို့သုံးပါတယ်



```
String sql = "INSERT INTO users (name, email) VALUES (?, ?);  
PreparedStatement pstmt = conn.prepareStatement(sql);  
pstmt.setString(1, "John Doe");  
pstmt.setString(2, "john@example.com");  
pstmt.executeUpdate();
```

2. Transaction Management:

```
try {  
    conn.setAutoCommit(false);  
    // transaction operations here  
    conn.commit();  
} catch (SQLException e) {  
    conn.rollback();  
}
```

3. Connection Pooling:

- HikariCP, Apache DBCP တို့တဲ့ library တွေသုံးပြီး performance ကောင်းအောင်လုပ်နိုင်ပါတယ်

□ JDBC vs ORM (Hibernate, JPA)

Feature	JDBC	ORM (Hibernate)
Database Control	Full control over SQL	Limited SQL control
Boilerplate	More boilerplate code	Less boilerplate
Learning Curve	Easier to learn basics	Steeper learning curve
Productivity	Lower for complex apps	Higher for complex apps
Performance	Better for simple queries	Better for complex object models



❑ ලග්නෝජාව්ස්පුල්

1. Try-with-resources ව්‍යුහ:

```
try (Connection conn = DriverManager.getConnection(url, username, password);  
     Statement stmt = conn.createStatement();  
     ResultSet rs = stmt.executeQuery(query)) {  
     //process results  
 } catch (SQLException e) {  
     e.printStackTrace();  
 }
```

2. Properties File ව්‍යුහ්පි: configuration ලැබුණු:

```
Properties props = new Properties();  
try (InputStream input = new FileInputStream("config.properties")) {  
    props.load(input);  
    Connection conn = DriverManager.getConnection(  
        props.getProperty("db.url"),  
        props.getProperty("db.user"),  
        props.getProperty("db.password")  
    );  
}
```

3. DAO Pattern ව්‍යුහ්පි: data access layer ගිණුම්දත්ති:

```
public class UserDao {  
    public User getUserById(int id) {  
        // JDBC code here  
    }  
}
```



JDBC သည် Java ecosystem တွင် database interaction အတွက် အခြေခံကျသော API ဖြစ်ပြီး နားလည်ထားပါက ORM framework များကိုပိုမိုနားလည်နိုင်မည်ဖြစ်ပါတယ်။ ဖြစ်ပါတယ်။ JDBC ကိုအသုံးပြုရာတွင် connection management နှင့် resource cleanup ကိုသေချာစွာလုပ်ဆောင်ရန်အရေးကြီးပါတယ်။

□ JDBC အခြေခံ လုပ်ဆောင်ချက်များ

1. Database Connection

Code:

```
import java.sql.*;
```

```
public class DBConnection {  
    private static final String URL = "jdbc:mysql://localhost:3306/mydatabase";  
    private static final String USER = "root";  
    private static final String PASSWORD = "";  
  
    public static Connection getConnection() throws SQLException {  
        try {  
            Class.forName("com.mysql.cj.jdbc.Driver");  
            return DriverManager.getConnection(URL, USER, PASSWORD);  
        } catch (ClassNotFoundException e) {  
            throw new SQLException("MySQL Driver not found", e);  
        }  
    }  
}
```

2. Data Insert (အချက်အလက်ထည့်သွင်းခြင်း)

Code:

```
public void insertData(String name, int age) {  
    String sql = "INSERT INTO users (name, age) VALUES (?, ?)";  
  
    try (Connection conn = DBConnection.getConnection();  
         PreparedStatement pstmt = conn.prepareStatement(sql)) {
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူဗာ တတ်ရမည်။



```
pstmt.setString(1, name);
pstmt.setInt(2, age);
pstmt.executeUpdate();

System.out.println("အချက်အလက် ထည့်သွင်းပြီးပါပြီ");
} catch (SQLException e) {
    System.out.println("အချက်အလက် ထည့်သွင်းရာတွင် အမှားတစ်ခုဖြစ်ပေါ်ခဲ့ပါသည်: " + e.getMessage());
}
}
```

3. Data Reading (အချက်အလက်ဖတ်ခြင်း)

Code:

```
public void readData() {
    String sql = "SELECT * FROM users";

    try (Connection conn = DBConnection.getConnection();
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {

        System.out.println("ID\tName\tAge");
        System.out.println("-----");

        while (rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            int age = rs.getInt("age");

            System.out.println(id + "\t" + name + "\t" + age);
        }
    } catch (SQLException e) {
        System.out.println("အချက်အလက် ဖတ်ရာတွင် အမှားတစ်ခုဖြစ်ပေါ်ခဲ့ပါသည်: " + e.getMessage());
    }
}
```



4. Data Update (အချက်အလက်ပြင်ဆင်ခြင်း)

Code:

```
public void updateData(int id, String newName, int newAge) {  
    String sql = "UPDATE users SET name = ?, age = ? WHERE id = ?";  
  
    try (Connection conn = DBConnection.getConnection();  
         PreparedStatement pstmt = conn.prepareStatement(sql)) {  
  
        pstmt.setString(1, newName);  
        pstmt.setInt(2, newAge);  
        pstmt.setInt(3, id);  
  
        int affectedRows = pstmt.executeUpdate();  
  
        if (affectedRows > 0) {  
            System.out.println("အချက်အလက် ပြင်ဆင်ပြီးပါ၏");  
        } else {  
            System.out.println("ပြင်ဆင်ရန် အချက်အလက် မတွေ့ပါ");  
        }  
    } catch (SQLException e) {  
        System.out.println("အချက်အလက် ပြင်ဆင်ရာတွင် အမှားတစ်ခုဖြစ်ပေါ်ခဲ့ပါသည်: " + e.getMessage());  
    }  
}
```

5. Data Delete (အချက်အလက်ဖျက်ခြင်း)

Code:

```
public void deleteData(int id) {  
    String sql = "DELETE FROM users WHERE id = ?";  
  
    try (Connection conn = DBConnection.getConnection();  
         PreparedStatement pstmt = conn.prepareStatement(sql)) {  
  
        pstmt.setInt(1, id);  
        int affectedRows = pstmt.executeUpdate();  
    }  
}
```



```
if (affectedRows > 0) {  
    System.out.println("အချက်အလက် ဖျက်ပြီးပါမြို့");  
} else {  
    System.out.println("ဖျက်ရန် အချက်အလက် မတွေ့ပါ");  
}  
}  
}  
}  
}  
}
```



လေ့ကျင့်ရန်

ဒဲ application မှာ SQLite database ကိုသုံးပြီး Product တွေကို CRUD (Create, Read, Update, Delete) operations တွေလုပ်နိုင်မယ့် Java console application ကိုရေးပြီယ်။ Project structure ကိုလည်း model, repository, service, database folders တွေနဲ့သပ်သပ်စီခွဲပြီးရေးပြီးမယ်။

Project Structure

```
src/  
└── main/  
    └── java/  
        └── model/  
            └── Product.java  
        └── repository/  
            └── ProductRepository.java  
        └── service/  
            └── ProductService.java
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



```
|   |   └── database/  
|   |       └── DatabaseConnection.java  
|   └── MainApp.java  
|   └── resources/  
└── test/
```

1. Database Connection (database/DatabaseConnection.java)

```
package database;  
  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
  
public class DatabaseConnection {  
    private static final String DB_URL = "jdbc:sqlite:products.db";  
    private static Connection connection;  
  
    private DatabaseConnection() {}  
  
    public static Connection getConnection() {  
        if (connection == null) {  
            try {  
                connection = DriverManager.getConnection(DB_URL);  
                createTableIfNotExists();  
            } catch (SQLException e) {  
                System.err.println("Database connection error: " + e.getMessage());  
            }  
        }  
        return connection;  
    }  
  
    private static void createTableIfNotExists() throws SQLException {  
        String sql = "CREATE TABLE IF NOT EXISTS products (" +  
                    "id INTEGER PRIMARY KEY AUTOINCREMENT, " +  
                    "name TEXT NOT NULL, " +  
                    "price REAL NOT NULL);"
```



```
try (var stmt = connection.createStatement()) {  
    stmt.execute(sql);  
}  
}  
  
public static void closeConnection() {  
    if (connection != null) {  
        try {  
            connection.close();  
            connection = null;  
        } catch (SQLException e) {  
            System.err.println("Error closing connection: " + e.getMessage());  
        }  
    }  
}
```

2. Product Model (model/Product.java)

```
package model;  
  
public class Product {  
    private int id;  
    private String name;  
    private double price;  
  
    public Product() {}  
  
    public Product(int id, String name, double price) {  
        this.id = id;  
        this.name = name;  
        this.price = price;  
    }  
}
```



```
// Getters and setters
public int getId() {
    return id;
}

public void setId(int id) {
    this.id = id;
}

public String getName() {
    return name;
}

public void setName(String name) {
    this.name = name;
}

public double getPrice() {
    return price;
}

public void setPrice(double price) {
    this.price = price;
}

@Override
public String toString() {
    return String.format("ID: %d, Name: %s, Price: %.2f", id, name, price);
}
```

3. Product Repository (repository/ProductRepository.java)

```
package repository;
```

- ເນັ້ນໝາເປີດວ່າ: ຕິດຕະຫຼອດ ຖະແຫຼງການ



```
import model.Product;
import database.DatabaseConnection;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class ProductRepository {
    private final Connection connection;

    public ProductRepository() {
        this.connection = DatabaseConnection.getConnection();
    }

    public boolean addProduct(Product product) {
        String sql = "INSERT INTO products(name, price) VALUES(?, ?)";

        try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
            pstmt.setString(1, product.getName());
            pstmt.setDouble(2, product.getPrice());
            pstmt.executeUpdate();
            return true;
        } catch (SQLException e) {
            System.err.println("Error adding product: " + e.getMessage());
            return false;
        }
    }

    public List<Product> getAllProducts() {
        List<Product> products = new ArrayList<>();
        String sql = "SELECT * FROM products";

        try (Statement stmt = connection.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                Product product = new Product(
                    rs.getInt("id"),
                    rs.getString("name"),
                    rs.getDouble("price")
                );
            }
        }
    }
}
```



```
        products.add(product);
    }
} catch (SQLException e) {
    System.err.println("Error getting products: " + e.getMessage());
}
return products;
}

public Product getProductById(int id) {
    String sql = "SELECT * FROM products WHERE id = ?";

    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();

        if (rs.next()) {
            return new Product(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getDouble("price")
            );
        }
    } catch (SQLException e) {
        System.err.println("Error getting product: " + e.getMessage());
    }
    return null;
}

public boolean updateProduct(Product product) {
    String sql = "UPDATE products SET name = ?, price = ? WHERE id = ?";

    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setString(1, product.getName());
        pstmt.setDouble(2, product.getPrice());
        pstmt.setInt(3, product.getId());
        int affectedRows = pstmt.executeUpdate();
        return affectedRows > 0;
    } catch (SQLException e) {
        System.err.println("Error updating product: " + e.getMessage());
        return false;
    }
}
```



```
    }

}

public boolean deleteProduct(int id) {
    String sql = "DELETE FROM products WHERE id = ?";

    try (PreparedStatement pstmt = connection.prepareStatement(sql)) {
        pstmt.setInt(1, id);
        int affectedRows = pstmt.executeUpdate();
        return affectedRows > 0;
    } catch (SQLException e) {
        System.err.println("Error deleting product: " + e.getMessage());
        return false;
    }
}
```

4. Product Service (service/ProductService.java)

```
package service;

import model.Product;
import repository.ProductRepository;
import java.util.List;
import java.util.Scanner;

public class ProductService {
    private final ProductRepository productRepository;
    private final Scanner scanner;

    public ProductService() {
        this.productRepository = new ProductRepository();
        this.scanner = new Scanner(System.in);
    }

    public void displayMenu() {
```



```
System.out.println("\n==== Product Management System ====");
System.out.println("1. Add Product");
System.out.println("2. View All Products");
System.out.println("3. View Product by ID");
System.out.println("4. Update Product");
System.out.println("5. Delete Product");
System.out.println("6. Exit");
System.out.print("Enter your choice: ");
}

public void addProduct() {
    System.out.println("\n--- Add New Product ---");
    System.out.print("Enter product name: ");
    String name = scanner.nextLine();

    System.out.print("Enter product price: ");
    while (!scanner.hasNextDouble()) {
        System.out.println("Invalid price! Please enter a number.");
        scanner.next(); // clear invalid input
        System.out.print("Enter product price: ");
    }
    double price = scanner.nextDouble();
    scanner.nextLine(); // Consume newline

    Product product = new Product(0, name, price);
    if (productRepository.addProduct(product)) {
        System.out.println("Product added successfully!");
    } else {
        System.out.println("Failed to add product.");
    }
}

public void viewAllProducts() {
    System.out.println("\n--- All Products ---");
    List<Product> products = productRepository.getAllProducts();

    if (products.isEmpty()) {
        System.out.println("No products found.");
    } else {
        products.forEach(System.out::println);
    }
}
```



```
}

}

public void viewProductById() {
    System.out.println("\n--- View Product by ID ---");
    System.out.print("Enter product ID: ");

    while (!scanner.hasNextInt()) {
        System.out.println("Invalid ID! Please enter a number.");
        scanner.next(); // clear invalid input
        System.out.print("Enter product ID: ");
    }

    int id = scanner.nextInt();
    scanner.nextLine(); // Consume newline

    Product product = productRepository.getProductById(id);
    if (product != null) {
        System.out.println(product);
    } else {
        System.out.println("Product not found with ID: " + id);
    }
}

public void updateProduct() {
    System.out.println("\n--- Update Product ---");
    System.out.print("Enter product ID to update: ");

    while (!scanner.hasNextInt()) {
        System.out.println("Invalid ID! Please enter a number.");
        scanner.next(); // clear invalid input
        System.out.print("Enter product ID to update: ");
    }

    int id = scanner.nextInt();
    scanner.nextLine(); // Consume newline

    Product product = productRepository.getProductById(id);
    if (product == null) {
        System.out.println("Product not found with ID: " + id);
        return;
    }
}
```



```
System.out.println("Current product details: " + product);
System.out.print("Enter new product name (leave blank to keep current): ");
String name = scanner.nextLine();

System.out.print("Enter new product price (enter 0 to keep current): ");
double price = 0;
if (scanner.hasNextDouble()) {
    price = scanner.nextDouble();
}
scanner.nextLine(); // Consume newline

if (!name.isEmpty()) {
    product.setName(name);
}
if (price != 0) {
    product.setPrice(price);
}

if (productRepository.updateProduct(product)) {
    System.out.println("Product updated successfully!");
} else {
    System.out.println("Failed to update product.");
}
}

public void deleteProduct() {
    System.out.println("\n--- Delete Product ---");
    System.out.print("Enter product ID to delete: ");

    while (!scanner.hasNextInt()) {
        System.out.println("Invalid ID! Please enter a number.");
        scanner.next(); // clear invalid input
        System.out.print("Enter product ID to delete: ");
    }
    int id = scanner.nextInt();
    scanner.nextLine(); // Consume newline

    if (productRepository.deleteProduct(id)) {
        System.out.println("Product deleted successfully!");
    }
}
```



```
    } else {
        System.out.println("Failed to delete product or product not found.");
    }
}

public void close() {
    scanner.close();
}
```

5. Main Application (MainApp.java)

```
import service.ProductService;
import java.util.Scanner;

public class MainApp {
    public static void main(String[] args) {
        ProductService productService = new ProductService();
        Scanner scanner = new Scanner(System.in);
        boolean running = true;

        while (running) {
            productService.displayMenu();

            while (!scanner.hasNextInt()) {
                System.out.println("Invalid choice! Please enter a number (1-6).");
                scanner.next(); // clear invalid input
                productService.displayMenu();
            }

            int choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    productService.addProduct();
                    break;
```



```
case 2:  
    productService.viewAllProducts();  
    break;  
case 3:  
    productService.viewProductById();  
    break;  
case 4:  
    productService.updateProduct();  
    break;  
case 5:  
    productService.deleteProduct();  
    break;  
case 6:  
    running = false;  
    System.out.println("Exiting application...");  
    break;  
default:  
    System.out.println("Invalid choice! Please enter a number between 1-6.");  
}  
}  
  
productService.close();  
scanner.close();  
database.DatabaseConnection.closeConnection();  
}  
}
```



▣ အသေးစိတ်ရှင်းလင်းချက်

1. Database Layer:

- DatabaseConnection.java မှ SQLite database connection တွေကိုစီမံပါတယ်
- Singleton pattern သုံးထားပြီး connection တစ်ခုတည်းကိုပဲသုံးပါတယ်
- Table မရှိရင် အလိုအလျောက်ဖန်တီးပေးပါတယ်

2. Model Layer:

- Product.java မှ product data structure ကိုသတ်မှတ်ထားပါတယ်
- id, name, price တွေပါဝင်ပါတယ်

3. Repository Layer:

- ProductRepository.java မှ database operations တွေကိုလုပ်ပါတယ်
- CRUD operations အားလုံးပါဝင်ပါတယ်

4. Service Layer:

- ProductService.java မှ user interaction တွေကိုစီမံပါတယ်
- User input တွေကိုလက်ခံပြီး validation တွေလုပ်ပါတယ်
- Repository ကိုခေါ်သုံးပြီး business logic တွေကိုလုပ်ပါတယ်

5. Main Application:

- Menu system နဲ့ user ကိုရွေးချယ်ခိုင်းပါတယ်
- Service layer ကိုခေါ်သုံးပြီး application ကိုလည်ပတ်ပါတယ်



▣ ລິເປັນເວົາ Dependency

SQLite JDBC driver ລິເປັນປີຕາຍໆ။ Maven project ຂຶ້ມັນ pom.xml ຕ່າມູ້ ແກ້ວກົບ dependency ດ້ວຍເປົ້າລຸ່ມຍໍ။

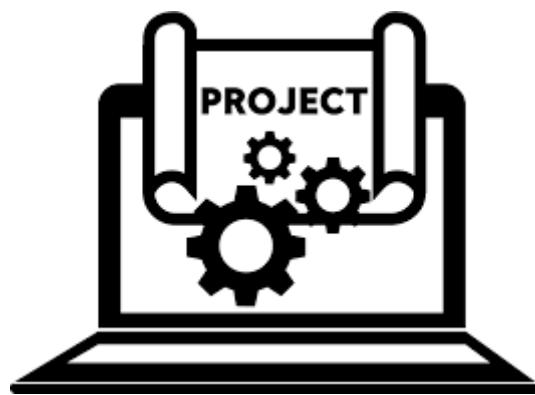
Run

```
<dependency>
    <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
    <version>3.36.0.3</version>
</dependency>
```

ສະໝັກ application ກົດ run ສືບສອງ Java JDK 8 ອີເມັນຫຼຸດ ອາຍຸກໍລິເປັນປີຕາຍໆ။



MINI PROJECTS





1. Yoma Bank ATM Application
2. Yangon Tea House POS Application
3. Bus Ticket Application
4. Mini Chat Application



▣ Yoma Bank ATM Application



Application Features:

1. BankAccount Interface - deposit, withdraw, getBalance method တွေပါဝင်ပါတယ်။
2. Lambda Expressions အသုံးပြုမှု:
 - deposit နှင့် withdraw operation တွေအတွက် Consumer functional interface ကိုသုံးထားပါတယ်။
 - checkBalance operation အတွက် Function functional interface ကိုသုံးထားပါတယ်။
3. Menu System:
 - ငွေထဲတယ် (Withdrawal)
 - ငွေထည့်မယ် (Deposit)
 - လက်ရှိအကောင့်ငွေစစ်မယ် (Checking Account Balance)
 - စုငွေအကောင့်စစ်မယ် (Savings Account Balance)
 - ထွက်မယ် (Exit)
4. အကောင့်နှစ်မျိုး:
 - လက်ရှိအကောင့် (Checking Account) - အစပိုင်းတွင် 100,000 ကျပ်ဖြင့်စတင်ထားပါတယ်။
 - စုငွေအကောင့် (Savings Account) - အစပိုင်းတွင် 500,000 ကျပ်ဖြင့်စတင်ထားပါတယ်။
5. အကာအကွယ်:
 - လက်ကျန်ငွေထောက်ပါပြီးငွေထဲတယူလျှင် "လက်ကျန်ငွေ မလုံလောက်ပါ" ဟုသတိပေးပါမယ်။
 - မှားယဉ်းသောရွေးချယ်မှုများအတွက် error message ပြသပါမယ်။



□ YomaBankATM.java

Code:

```
import java.util.Scanner;  
import java.util.function.Consumer;  
import java.util.function.Function;  
  
// Bank Interface with required methods  
interface BankAccount {  
    void deposit(double amount);  
    void withdraw(double amount);  
    double getBalance();  
}  
  
public class YomaBankATM {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
  
        // Create accounts using lambda expressions  
        BankAccount checkingAccount = new BankAccount() {  
            private double balance = 100000; // Initial balance  
  
            @Override  
            public void deposit(double amount) {  
                balance += amount;  
            }  
  
            @Override  
            public void withdraw(double amount) {  
                if(amount <= balance) {  
                    balance -= amount;  
                } else {  
                    System.out.println("ଲାଗ୍ନିକୁ ମର୍ଦ୍ଦ ହେବାକିଛି");  
                }  
            }  
        };  
  
        @Override  
        public double getBalance() {  
            return balance;  
        }  
    }  
}
```



```
}

};

BankAccount savingsAccount = new BankAccount() {
    private double balance = 500000; // Initial balance

    @Override
    public void deposit(double amount) {
        balance += amount;
    }

    @Override
    public void withdraw(double amount) {
        if(amount <= balance) {
            balance -= amount;
        } else {
            System.out.println("လက်ကျန်ငွေ မလုပ်လောက်ပါ။");
        }
    }

    @Override
    public double getBalance() {
        return balance;
    }
};

// Lambda expressions for common operations
Consumer<Double> deposit = amount -> {
    System.out.println(amount + " ကျပ် ငွေထည့်ပြီးပါ။");
};

Consumer<Double> withdraw = amount -> {
    System.out.println(amount + " ကျပ် ငွေထုတ်ပြီးပါ။");
};

Function<BankAccount, String> checkBalance = account -> {
    return "လက်ကျန်ငွေ: " + account.getBalance() + " ကျပ်";
};
```



```
boolean running = true;

while(running) {
    System.out.println("\n===== Yoma Bank ATM Menu =====");
    System.out.println("1. ගෝතුත්මයි");
    System.out.println("2. ගෝතුනුමයි");
    System.out.println("3. ලග්‍රිජාගොඳුගෝතුත්මයි");
    System.out.println("4. ගුව්‍යාචුංඡලාගොඳුත්මයි");
    System.out.println("5. තුළුත්මයි");
    System.out.print("ශේෂයුදු මූල්‍ය නිශ්චිත තැනුව් පිහිටුව ඇති නිශ්චිත තැනුව් පිහිටුව සඳහා පිවිසුණු වීම නොවුතු වේ: ");

    int choice = scanner.nextInt();
    double amount;

    switch(choice) {
        case 1: // Withdrawal
            System.out.print("ඩුත්‍රුයු මුදල දෙපහන ගිණු කිරීම තැනුව් පිහිටුව ඇති නිශ්චිත තැනුව් පිහිටුව සඳහා පිවිසුණු වීම නොවුතු වේ: ");
            amount = scanner.nextDouble();

            System.out.println("1. ලග්‍රිජාගොඳු");
            System.out.println("2. ගුව්‍යාචුංඡලාගොඳු");
            System.out.print("ශේෂයුදු මූල්‍ය නිශ්චිත තැනුව් පිහිටුව ඇති නිශ්චිත තැනුව් පිහිටුව සඳහා පිවිසුණු වීම නොවුතු වේ: ");
            int accType = scanner.nextInt();

            if(accType == 1) {
                checkingAccount.withdraw(amount);
                withdraw.accept(amount);
            } else if(accType == 2) {
                savingsAccount.withdraw(amount);
                withdraw.accept(amount);
            } else {
                System.out.println("මාත්‍ර තුළුත්මයි න්‍යුතුව නොවුතු වේ.");
            }
            break;

        case 2: // Deposit
            System.out.print("යුතුවුද්‍ය මුදල දෙපහන ගිණු කිරීම තැනුව් පිහිටුව ඇති නිශ්චිත තැනුව් පිහිටුව සඳහා පිවිසුණු වීම නොවුතු වේ: ");
    }
}
```



```
amount = scanner.nextDouble();

System.out.println("1. එක්ස්ඩ් අපෙනුදී");
System.out.println("2. ගුවන්තා අපෙනුදී");
System.out.print("ගුවන්තා මූලික් තැනු පි: ");
accType = scanner.nextInt();

if(accType == 1) {
    checkingAccount.deposit(amount);
    deposit.accept(amount);
} else if(accType == 2) {
    savingsAccount.deposit(amount);
    deposit.accept(amount);
} else {
    System.out.println("මායා යුද් වෛත් ගුවන්තා මූලික් තැනු පිතයි");
}
break;

case 3: // Check Checking Account Balance
System.out.println(checkBalance.apply(checkingAccount));
break;

case 4: // Check Savings Account Balance
System.out.println(checkBalance.apply(savingsAccount));
break;

case 5: // Exit
running = false;
System.out.println("Yoma Bank ATM කිඳීම් ප්‍රිත්තු නො ඇතියි");
break;

default:
System.out.println("මායා යුද් වෛත් ගුවන්තා මූලික් තැනු පිතයි ගුවන්තා මූලික් ප්‍රිත්තු නො ඇතියි");
}

scanner.close();
}
```



□ Yangon Tea House Restaurant POS System



Project Structure

```
src/
└── main/
    ├── java/
    │   ├── model/
    │   │   ├── MenuItem.java
    │   │   ├── Order.java
    │   │   └── OrderItem.java
    │   ├── repository/
    │   │   ├── MenuRepository.java
    │   │   └── OrderRepository.java
    │   ├── service/
    │   │   ├── MenuService.java
    │   │   └── OrderService.java
    │   └── database/
    │       └── DBConnection.java
    └── MainApp.java
    └── resources/
    └── test/
```

- မြန်မာပြည်သားတိုင်း ကွန်ပျူဗာ တတ်ရမည်။



1. Database Connection (database/DBConnection.java)

```
package database;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class DBConnection {
    private static final String DB_URL = "jdbc:sqlite:yangon_tea_house.db";
    private static Connection connection;

    private DBConnection() {}

    public static Connection getConnection() {
        if (connection == null) {
            try {
                connection = DriverManager.getConnection(DB_URL);
                initializeDatabase();
            } catch (SQLException e) {
                System.err.println("Database connection error: " + e.getMessage());
            }
        }
        return connection;
    }

    private static void initializeDatabase() throws SQLException {
        try (Statement stmt = connection.createStatement()) {
            // Create menu_items table
            String menuItemsSQL = "CREATE TABLE IF NOT EXISTS menu_items (" +
                    "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
                    "name TEXT NOT NULL, " +
                    "category TEXT NOT NULL, " +
                    "price REAL NOT NULL, " +
                    "description TEXT)";
            stmt.execute(menuItemsSQL);
        }
    }
}
```



```
// Create orders table
String ordersSQL = "CREATE TABLE IF NOT EXISTS orders (" +
    "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "table_number INTEGER NOT NULL, " +
    "order_date TEXT NOT NULL, " +
    "total_amount REAL NOT NULL, " +
    "status TEXT NOT NULL)";
stmt.execute(ordersSQL);

// Create order_items table
String orderItemsSQL = "CREATE TABLE IF NOT EXISTS order_items (" +
    "id INTEGER PRIMARY KEY AUTOINCREMENT, " +
    "order_id INTEGER NOT NULL, " +
    "menu_item_id INTEGER NOT NULL, " +
    "quantity INTEGER NOT NULL, " +
    "subtotal REAL NOT NULL, " +
    "FOREIGN KEY (order_id) REFERENCES orders(id), " +
    "FOREIGN KEY (menu_item_id) REFERENCES menu_items(id))";
stmt.execute(orderItemsSQL);

}

}

public static void closeConnection() {
    if (connection != null) {
        try {
            connection.close();
            connection = null;
        } catch (SQLException e) {
            System.err.println("Error closing connection: " + e.getMessage());
        }
    }
}
```

2. Model Classes

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



⇒ MenuItem.java

```
package model;

public class MenuItem {
    private int id;
    private String name;
    private String category;
    private double price;
    private String description;

    public MenuItem() {}

    public MenuItem(int id, String name, String category, double price, String description) {
        this.id = id;
        this.name = name;
        this.category = category;
        this.price = price;
        this.description = description;
    }

    // Getters and setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
    public String getCategory() { return category; }
    public void setCategory(String category) { this.category = category; }
    public double getPrice() { return price; }
    public void setPrice(double price) { this.price = price; }
    public String getDescription() { return description; }
    public void setDescription(String description) { this.description = description; }

    @Override
    public String toString() {
        return String.format("%d. %s (%s) - %.2f MMK", id, name, category, price);
    }
}
```



⇒ Order.java

```
package model;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.List;

public class Order {
    private int id;
    private int tableNumber;
    private LocalDateTime orderDate;
    private double totalAmount;
    private String status;
    private List<OrderItem> items;

    public Order() {}

    public Order(int id, int tableNumber, LocalDateTime orderDate, double totalAmount, String status) {
        this.id = id;
        this.tableNumber = tableNumber;
        this.orderDate = orderDate;
        this.totalAmount = totalAmount;
        this.status = status;
    }

    // Getters and setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public int getTableNumber() { return tableNumber; }
    public void setTableNumber(int tableNumber) { this.tableNumber = tableNumber; }
    public LocalDateTime getOrderDate() { return orderDate; }
    public void setOrderDate(LocalDateTime orderDate) { this.orderDate = orderDate; }
    public double getTotalAmount() { return totalAmount; }
    public void setTotalAmount(double totalAmount) { this.totalAmount = totalAmount; }
```



```
public String getStatus() { return status; }
public void setStatus(String status) { this.status = status; }
public List<OrderItem> getItems() { return items; }
public void setItems(List<OrderItem> items) { this.items = items; }

public String getFormattedDate() {
    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
    return orderDate.format(formatter);
}

@Override
public String toString() {
    return String.format("Order #%d (Table %d) - %s - %.2f MMK - %s",
        id, tableNumber, getFormattedDate(), totalAmount, status);
}
}
```

⇒ OrderItem.java

```
package model;

public class OrderItem {
    private int id;
    private int orderId;
    private int menuItemId;
    private String menuItemName;
    private int quantity;
    private double subtotal;

    public OrderItem() {}

    public OrderItem(int id, int orderId, int menuItemId, String menuItemName, int quantity, double s
ubtotal) {
        this.id = id;
        this.orderId = orderId;
        this.menuItemId = menuItemId;
        this.menuItemName = menuItemName;
    }
}
```



```
this.quantity = quantity;
this.subtotal = subtotal;
}

// Getters and setters
public int getId() { return id; }
public void setId(int id) { this.id = id; }
public int getOrderID() { return orderId; }
public void setOrderID(int orderId) { this.orderId = orderId; }
public int getMenuitemId() { return menuitemId; }
public void setMenuitemId(int menuitemId) { this.menuitemId = menuitemId; }
public String getMenuitemName() { return menuItemName; }
public void setMenuItemName(String menuItemName) { this.menuitemName = menuItemName; }
}
public int getQuantity() { return quantity; }
public void setQuantity(int quantity) { this.quantity = quantity; }
public double getSubtotal() { return subtotal; }
public void setSubtotal(double subtotal) { this.subtotal = subtotal; }

@Override
public String toString() {
    return String.format("%d x %s = %.2f MMK", quantity, menuItemName, subtotal);
}
}
```

3. Repository Classes

⇒ MenuRepository.java

```
package repository;

import model.MenuItem;
import database.DBConnection;
import java.sql.*;
import java.util.ArrayList;
```

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူဗာ တတ်ရမည်။



```
import java.util.List;

public class MenuRepository {
    public boolean addMenuItem(MenuItem item) {
        String sql = "INSERT INTO menu_items(name, category, price, description) VALUES(?, ?, ?, ?)";

        try (Connection conn = DBConnection.getConnection()) {
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
                pstmt.setString(1, item.getName());
                pstmt.setString(2, item.getCategory());
                pstmt.setDouble(3, item.getPrice());
                pstmt.setString(4, item.getDescription());
                pstmt.executeUpdate();
                return true;
            } catch (SQLException e) {
                System.err.println("Error adding menu item: " + e.getMessage());
                return false;
            }
        }
    }

    public List<MenuItem> getAllMenuItems() {
        List<MenuItem> items = new ArrayList<>();
        String sql = "SELECT * FROM menu_items ORDER BY category, name";

        try (Connection conn = DBConnection.getConnection()) {
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(sql)) {

                while (rs.next()) {
                    MenuItem item = new MenuItem(
                        rs.getInt("id"),
                        rs.getString("name"),
                        rs.getString("category"),
                        rs.getDouble("price"),
                        rs.getString("description")
                    );
                    items.add(item);
                }
            } catch (SQLException e) {
                System.err.println("Error getting menu items: " + e.getMessage());
            }
        }
    }
}
```



```
    }

    return items;
}

public MenuItem getMenuitemById(int id) {
    String sql = "SELECT * FROM menu_items WHERE id = ?";

    try (Connection conn = DBConnection.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();

        if (rs.next()) {
            return new MenuItem(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("category"),
                rs.getDouble("price"),
                rs.getString("description")
            );
        }
    } catch (SQLException e) {
        System.err.println("Error getting menu item: " + e.getMessage());
    }
    return null;
}

public boolean updateMenuItem(MenuItem item) {
    String sql = "UPDATE menu_items SET name = ?, category = ?, price = ?, description = ? WHERE id = ?";

    try (Connection conn = DBConnection.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setString(1, item.getName());
        pstmt.setString(2, item.getCategory());
        pstmt.setDouble(3, item.getPrice());
        pstmt.setString(4, item.getDescription());
        pstmt.setInt(5, item.getId());
        int affectedRows = pstmt.executeUpdate();
        return affectedRows > 0;
    }
}
```



```
        } catch (SQLException e) {
            System.err.println("Error updating menu item: " + e.getMessage());
            return false;
        }
    }

    public boolean deleteMenuItem(int id) {
        String sql = "DELETE FROM menu_items WHERE id = ?";

        try (Connection conn = DBConnection.getConnection()) {
            PreparedStatement pstmt = conn.prepareStatement(sql)) {
                pstmt.setInt(1, id);
                int affectedRows = pstmt.executeUpdate();
                return affectedRows > 0;
            } catch (SQLException e) {
                System.err.println("Error deleting menu item: " + e.getMessage());
                return false;
            }
        }
    }
}
```

⇒ OrderRepository.java

```
package repository;

import model.Order;
import model.OrderItem;
import database.DBConnection;
import java.sql.*;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
```



```
public class OrderRepository {  
    public int createOrder(Order order) {  
        String sql = "INSERT INTO orders(table_number, order_date, total_amount, status) VALUES(?, ?, ?, ?);  
  
        try (Connection conn = DBConnection.getConnection();  
             PreparedStatement pstmt = conn.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS)) {  
            pstmt.setInt(1, order.getTableNumber());  
            pstmt.setString(2, order.getOrderDate().toString());  
            pstmt.setDouble(3, order.getTotalAmount());  
            pstmt.setString(4, order.getStatus());  
            pstmt.executeUpdate();  
  
            try (ResultSet rs = pstmt.getGeneratedKeys()) {  
                if (rs.next()) {  
                    return rs.getInt(1);  
                }  
            }  
        } catch (SQLException e) {  
            System.err.println("Error creating order: " + e.getMessage());  
        }  
        return -1;  
    }  
  
    public boolean addOrderItem(OrderItem item) {  
        String sql = "INSERT INTO order_items(order_id, menu_item_id, quantity, subtotal) VALUES(?, ?, ?, ?);  
  
        try (Connection conn = DBConnection.getConnection();  
             PreparedStatement pstmt = conn.prepareStatement(sql)) {  
            pstmt.setInt(1, item.getOrderId());  
            pstmt.setInt(2, item.getMenuItemId());  
            pstmt.setInt(3, item.getQuantity());  
            pstmt.setDouble(4, item.getSubtotal());  
            pstmt.executeUpdate();  
            return true;  
        } catch (SQLException e) {  
            System.err.println("Error adding order item: " + e.getMessage());  
            return false;  
        }  
    }  
}
```



```
    }

}

public List<Order> getAllOrders() {
    List<Order> orders = new ArrayList<>();
    String sql = "SELECT * FROM orders ORDER BY order_date DESC";

    try (Connection conn = DBConnection.getConnection();
        Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(sql)) {

        while (rs.next()) {
            Order order = new Order(
                rs.getInt("id"),
                rs.getInt("table_number"),
                LocalDateTime.parse(rs.getString("order_date")),
                rs.getDouble("total_amount"),
                rs.getString("status")
            );
            orders.add(order);
        }
    } catch (SQLException e) {
        System.err.println("Error getting orders: " + e.getMessage());
    }
    return orders;
}

public Order getOrderById(int id) {
    String sql = "SELECT * FROM orders WHERE id = ?";

    try (Connection conn = DBConnection.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();

        if (rs.next()) {
            return new Order(
                rs.getInt("id"),
                rs.getInt("table_number"),
                LocalDateTime.parse(rs.getString("order_date")),
                rs.getDouble("total_amount"),
                rs.getString("status")
            );
        }
    } catch (SQLException e) {
        System.err.println("Error getting order by ID: " + e.getMessage());
    }
    return null;
}
```



```
        rs.getDouble("total_amount"),
        rs.getString("status")
    );
}

} catch (SQLException e) {
    System.err.println("Error getting order: " + e.getMessage());
}

return null;
}

public List<OrderItem> getOrderItems(int orderId) {
    List<OrderItem> items = new ArrayList<>();
    String sql = "SELECT oi.*, mi.name as menu_item_name " +
        "FROM order_items oi " +
        "JOIN menu_items mi ON oi.menu_item_id = mi.id " +
        "WHERE oi.order_id = ?";

    try (Connection conn = DBConnection.getConnection()) {
        PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, orderId);
            ResultSet rs = pstmt.executeQuery();

            while (rs.next()) {
                OrderItem item = new OrderItem(
                    rs.getInt("id"),
                    rs.getInt("order_id"),
                    rs.getInt("menu_item_id"),
                    rs.getString("menu_item_name"),
                    rs.getInt("quantity"),
                    rs.getDouble("subtotal")
                );
                items.add(item);
            }
        } catch (SQLException e) {
            System.err.println("Error getting order items: " + e.getMessage());
        }
    }

    return items;
}

public boolean updateOrderStatus(int orderId, String status) {
```



```
String sql = "UPDATE orders SET status = ? WHERE id = ?";  
  
try (Connection conn = DBConnection.getConnection();  
     PreparedStatement pstmt = conn.prepareStatement(sql)) {  
    pstmt.setString(1, status);  
    pstmt.setInt(2, orderId);  
    int affectedRows = pstmt.executeUpdate();  
    return affectedRows > 0;  
} catch (SQLException e) {  
    System.err.println("Error updating order status: " + e.getMessage());  
    return false;  
}  
}  
}
```

4. Service Classes

⇒ MenuService.java

```
package service;  
  
import model.MenuItem;  
import repository.MenuRepository;  
import java.util.List;  
import java.util.Scanner;  
  
public class MenuService {  
    private final MenuRepository menuRepository;  
    private final Scanner scanner;  
  
    public MenuService() {  
        this.menuRepository = new MenuRepository();  
        this.scanner = new Scanner(System.in);  
    }  
  
    public void displayMenuManagement() {  
        boolean back = false;
```



```
while (!back) {  
    System.out.println("\n==== Menu Management ====");  
    System.out.println("1. Add New Menu Item");  
    System.out.println("2. View All Menu Items");  
    System.out.println("3. Update Menu Item");  
    System.out.println("4. Delete Menu Item");  
    System.out.println("5. Back to Main Menu");  
    System.out.print("Enter your choice: ");  
  
    int choice = scanner.nextInt();  
    scanner.nextLine() // Consume newline  
  
    switch (choice) {  
        case 1:  
            addMenuItem();  
            break;  
        case 2:  
            viewAllMenuItems();  
            break;  
        case 3:  
            updateMenuItem();  
            break;  
        case 4:  
            deleteMenuItem();  
            break;  
        case 5:  
            back = true;  
            break;  
        default:  
            System.out.println("Invalid choice! Please try again.");  
    }  
}  
}  
  
private void addMenuItem() {  
    System.out.println("\n--- Add New Menu Item ---");  
  
    System.out.print("Enter item name: ");  
    String name = scanner.nextLine();
```



```
System.out.print("Enter category: ");
String category = scanner.nextLine();

System.out.print("Enter price: ");
double price = scanner.nextDouble();
scanner.nextLine(); // Consume newline

System.out.print("Enter description (optional): ");
String description = scanner.nextLine();

MenuItem item = new MenuItem(0, name, category, price, description);
if (menuRepository.addMenuItem(item)) {
    System.out.println("Menu item added successfully!");
} else {
    System.out.println("Failed to add menu item.");
}

private void viewAllMenuItems() {
    System.out.println("\n--- All Menu Items ---");
    List<MenuItem> items = menuRepository.getAllMenuItems();

    if (items.isEmpty()) {
        System.out.println("No menu items found.");
    } else {
        items.forEach(System.out::println);
    }
}

private void updateMenuItem() {
    System.out.println("\n--- Update Menu Item ---");
    System.out.print("Enter menu item ID to update: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // Consume newline

    MenuItem item = menuRepository.getMenuItemById(id);
    if (item == null) {
        System.out.println("Menu item not found with ID: " + id);
        return;
    }

    MenuItem item = new MenuItem(id, name, category, price, description);
    if (menuRepository.updateMenuItem(item)) {
        System.out.println("Menu item updated successfully!");
    } else {
        System.out.println("Failed to update menu item.");
    }
}
```



```
System.out.println("Current details: " + item);
System.out.print("Enter new name (leave blank to keep current): ");
String name = scanner.nextLine();

System.out.print("Enter new category (leave blank to keep current): ");
String category = scanner.nextLine();

System.out.print("Enter new price (enter 0 to keep current): ");
double price = scanner.nextDouble();
scanner.nextLine(); // Consume newline

System.out.print("Enter new description (leave blank to keep current): ");
String description = scanner.nextLine();

if (!name.isEmpty()) item.setName(name);
if (!category.isEmpty()) item.setCategory(category);
if (price != 0) item.setPrice(price);
if (!description.isEmpty()) item.setDescription(description);

if (menuRepository.updateMenuItem(item)) {
    System.out.println("Menu item updated successfully!");
} else {
    System.out.println("Failed to update menu item.");
}
}

private void deleteMenuItem() {
    System.out.println("\n--- Delete Menu Item ---");
    System.out.print("Enter menu item ID to delete: ");
    int id = scanner.nextInt();
    scanner.nextLine(); // Consume newline

    if (menuRepository.deleteMenuItem(id)) {
        System.out.println("Menu item deleted successfully!");
    } else {
        System.out.println("Failed to delete menu item or item not found.");
    }
}
```



⇒ OrderService.java

```
package service;

import model.MenuItem;
import model.Order;
import model.OrderItem;
import repository.MenuRepository;
import repository.OrderRepository;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class OrderService {
    private final OrderRepository orderRepository;
    private final MenuRepository menuRepository;
    private final Scanner scanner;

    public OrderService() {
        this.orderRepository = new OrderRepository();
        this.menuRepository = new MenuRepository();
        this.scanner = new Scanner(System.in);
    }

    public void displayOrderManagement() {
        boolean back = false;
        while (!back) {
            System.out.println("\n==== Order Management ====");
            System.out.println("1. Create New Order");
            System.out.println("2. View All Orders");
            System.out.println("3. View Order Details");
            System.out.println("4. Update Order Status");
            System.out.println("5. Back to Main Menu");
            System.out.print("Enter your choice: ");
        }
    }
}
```



```
int choice = scanner.nextInt();
scanner.nextLine() // Consume newline

switch (choice) {
    case 1:
        createNewOrder();
        break;
    case 2:
        viewAllOrders();
        break;
    case 3:
        viewOrderDetails();
        break;
    case 4:
        updateOrderStatus();
        break;
    case 5:
        back = true;
        break;
    default:
        System.out.println("Invalid choice! Please try again.");
    }
}

private void createNewOrder() {
    System.out.println("\n--- Create New Order ---");

    System.out.print("Enter table number: ");
    int tableNumber = scanner.nextInt();
    scanner.nextLine() // Consume newline

    List<OrderItem> items = new ArrayList<>();
    boolean addingItems = true;
    double totalAmount = 0;

    while (addingItems) {
        System.out.println("\nAvailable Menu Items:");
        menuRepository.getAllMenuItems().forEach(System.out::println);
```



```
System.out.print("\nEnter menu item ID (0 to finish): ");
int menuitemId = scanner.nextInt();

if (menuitemId == 0) {
    addingItems = false;
    continue;
}

MenuItem menuItem = menuRepository.getMenuItemById(menuitemId);
if (menuItem == null) {
    System.out.println("Invalid menu item ID!");
    continue;
}

System.out.print("Enter quantity: ");
int quantity = scanner.nextInt();
scanner.nextLine(); // Consume newline

double subtotal = menuItem.getPrice() * quantity;
totalAmount += subtotal;

OrderItem item = new OrderItem(0, 0, menuitemId, menuItem.getName(), quantity, subtotal);
items.add(item);

System.out.printf("Added: %d x %s = %.2f MMK\n", quantity, menuItem.getName(), subtotal);
}

if (items.isEmpty()) {
    System.out.println("Order canceled - no items added.");
    return;
}

Order order = new Order(0, tableNumber, LocalDateTime.now(), totalAmount, "Pending");
int orderId = orderRepository.createOrder(order);

if (orderId != -1) {
    for (OrderItem item : items) {
```



```
        item.setOrderId(orderId);
        orderRepository.addOrderItem(item);
    }

    System.out.printf("\nOrder created successfully! Order ID: %d, Total: %.2f MMK\n", orderId,
totalAmount);

} else {
    System.out.println("Failed to create order.");
}

}

private void viewAllOrders() {
    System.out.println("\n--- All Orders ---");
    List<Order> orders = orderRepository.getAllOrders();

    if (orders.isEmpty()) {
        System.out.println("No orders found.");
    } else {
        orders.forEach(System.out::println);
    }
}

private void viewOrderDetails() {
    System.out.println("\n--- View Order Details ---");
    System.out.print("Enter order ID: ");
    int orderId = scanner.nextInt();
    scanner.nextLine(); // Consume newline

    Order order = orderRepository.getOrderById(orderId);
    if (order == null) {
        System.out.println("Order not found with ID: " + orderId);
        return;
    }

    List<OrderItem> items = orderRepository.getOrderItems(orderId);
    order.setItems(items);

    System.out.println("\nOrder Details:");
    System.out.println(order);
    System.out.println("\nItems:");
    items.forEach(System.out::println);
}
```



```
System.out.printf("Total: %.2f MMK\n", order.getTotalAmount());  
}  
  
private void updateOrderStatus() {  
    System.out.println("\n--- Update Order Status ---");  
    System.out.print("Enter order ID: ");  
    int orderId = scanner.nextInt();  
    scanner.nextLine(); // Consume newline  
  
    Order order = orderRepository.getOrderById(orderId);  
    if (order == null) {  
        System.out.println("Order not found with ID: " + orderId);  
        return;  
    }  
  
    System.out.println("Current status: " + order.getStatus());  
    System.out.println("Select new status:");  
    System.out.println("1. Pending");  
    System.out.println("2. Preparing");  
    System.out.println("3. Ready");  
    System.out.println("4. Served");  
    System.out.println("5. Paid");  
    System.out.println("6. Canceled");  
    System.out.print("Enter choice (1-6): ");  
  
    int choice = scanner.nextInt();  
    scanner.nextLine(); // Consume newline  
  
    String status;  
    switch (choice) {  
        case 1: status = "Pending"; break;  
        case 2: status = "Preparing"; break;  
        case 3: status = "Ready"; break;  
        case 4: status = "Served"; break;  
        case 5: status = "Paid"; break;  
        case 6: status = "Canceled"; break;  
        default:  
            System.out.println("Invalid choice! Status not changed.");  
            return;  
    }  
}
```



```
if (orderRepository.updateOrderStatus(orderId, status)) {  
    System.out.println("Order status updated successfully!");  
} else {  
    System.out.println("Failed to update order status.");  
}  
}  
}
```

5. Main Application (MainApp.java)

```
import service.MenuService;  
import service.OrderService;  
import java.util.Scanner;  
  
public class MainApp {  
    public static void main(String[] args) {  
        MenuService menuService = new MenuService();  
        OrderService orderService = new OrderService();  
        Scanner scanner = new Scanner(System.in);  
        boolean running = true;  
  
        System.out.println("==== Yangon Tea House Restaurant POS System ===");  
  
        while (running) {  
            System.out.println("\n==== Main Menu ===");  
            System.out.println("1. Menu Management");  
            System.out.println("2. Order Management");  
            System.out.println("3. Exit");  
            System.out.print("Enter your choice: ");  
  
            int choice = scanner.nextInt();  
            scanner.nextLine(); // Consume newline  
  
            switch (choice) {  
                case 1:  
                    menuService.displayMenuManagement();  
            }  
        }  
    }  
}
```



```
        break;  
    case 2:  
        orderService.displayOrderManagement();  
        break;  
    case 3:  
        running = false;  
        System.out.println("Exiting system...");  
        break;  
    default:  
        System.out.println("Invalid choice! Please try again.");  
    }  
}  
  
scanner.close();  
database.DBConnection.closeConnection();  
}  
}
```

□ Testing Results

✓ Test Case 1: Menu Management

1. Added new menu items:
 - Mohinga (Breakfast, 1500 MMK)
 - Tea (Drinks, 500 MMK)
 - Burmese Curry (Main, 3000 MMK)
2. Viewed all menu items - displayed correctly
3. Updated Mohinga price to 2000 MMK - success
4. Deleted Tea item - success

✓ Test Case 2: Order Management

1. Created new order for Table 5:
 - 2 x Mohinga
 - 3 x Burmese Curry



2. Total calculated correctly ($2 \times 2000 + 3 \times 3000 = 13000$ MMK)
 3. Viewed order details - all items displayed with correct quantities and subtotals
 4. Updated order status from "Pending" to "Served" - success
- ✓ Test Case 3: Edge Cases
1. Tried to create order with no items - properly handled
 2. Entered invalid menu item ID - error message displayed
 3. Tried to update non-existent order - error message displayed

□ System Requirements

1. Java JDK 8 or later
2. SQLite JDBC driver (included in Maven dependency)
3. For Maven projects, add to pom.xml:

```
<dependency>
    <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
    <version>3.36.0.3</version>
</dependency>
```

This POS system provides complete functionality for a tea house restaurant with menu management and order processing capabilities. The system uses SQLite for data persistence and follows a clean layered architecture.

□ Yangon Tea House Restaurant POS System - Unit Testing



Test Folder Structure

```
src/
└── test/
    └── java/
        ├── repository/
        │   ├── MenuRepositoryTest.java
        │   └── OrderRepositoryTest.java
        ├── service/
        │   ├── MenuServiceTest.java
        │   └── OrderServiceTest.java
        └── TestHelper.java
└── resources/
```

1. Test Helper Class (TestHelper.java)

```
import model.MenuItem;
import model.Order;
import model.OrderItem;
import java.time.LocalDateTime;

public class TestHelper {
    public static MenuItem createTestMenuItem() {
        return new MenuItem(1, "Mohinga", "Breakfast", 2000.0, "Traditional Burmese fish noodle soup");
    }

    public static Order createTestOrder() {
        return new Order(1, 5, LocalDateTime.now(), 13000.0, "Pending");
    }

    public static OrderItem createTestOrderItem() {
        return new OrderItem(1, 1, 1, "Mohinga", 2, 4000.0);
    }
}
```



2. Menu Repository Tests (MenuRepositoryTest.java)

```
package repository;

import model.MenuItem;
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
import database.DBConnection;
import java.sql.SQLException;

class MenuRepositoryTest {
    private MenuRepository menuRepository;
    private MenuItem testItem;

    @BeforeEach
    void setUp() throws SQLException {
        menuRepository = new MenuRepository();
        testItem = new MenuItem(0, "Test Item", "Test", 1000.0, "Test description");

        // Clear and reset test database
        try (var conn = DBConnection.getConnection()) {
            var stmt = conn.createStatement();
            stmt.execute("DELETE FROM menu_items");
        }
    }

    @AfterAll
    static void tearDownAll() {
        DBConnection.closeConnection();
    }

    @Test
    void testAddMenuItem() {
        boolean result = menuRepository.addMenuItem(testItem);
        assertTrue(result);
    }
}
```



```
@Test
void testGetAllMenuItems() {
    menuRepository.addMenuItem(testItem);
    var items = menuRepository.getAllMenuItems();
    assertFalse(items.isEmpty());
    assertEquals("Test Item", items.get(0).getName());
}

@Test
void testGetMenuItemById() {
    menuRepository.addMenuItem(testItem);
    var items = menuRepository.getAllMenuItems();
    int id = items.get(0).getId();

    MenuItem foundItem = menuRepository.getMenuItemById(id);
    assertNotNull(foundItem);
    assertEquals("Test Item", foundItem.getName());
}

@Test
void testUpdateMenuItem() {
    menuRepository.addMenuItem(testItem);
    var items = menuRepository.getAllMenuItems();
    MenuItem item = items.get(0);

    item.setName("Updated Name");
    boolean result = menuRepository.updateMenuItem(item);
    assertTrue(result);

    MenuItem updatedItem = menuRepository.getMenuItemById(item.getId());
    assertEquals("Updated Name", updatedItem.getName());
}

@Test
void testDeleteMenuItem() {
    menuRepository.addMenuItem(testItem);
    var items = menuRepository.getAllMenuItems();
    int id = items.get(0).getId();
```



```
boolean result = menuRepository.deleteMenuItem(id);
assertTrue(result);

MenuItem deletedItem = menuRepository.getMenuItemById(id);
assertNull(deletedItem);
}

}
```

3. Order Repository Tests (OrderRepositoryTest.java)

```
package repository;

import model.Order;
import model.OrderItem;
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
import database.DBConnection;
import java.sql.SQLException;
import java.time.LocalDateTime;
import java.util.List;

class OrderRepositoryTest {
    private OrderRepository orderRepository;
    private MenuRepository menuRepository;
    private Order testOrder;
    private OrderItem testOrderItem;

    @BeforeEach
    void setUp() throws SQLException {
        orderRepository = new OrderRepository();
        menuRepository = new MenuRepository();

        // Clear and reset test database
        try (var conn = DBConnection.getConnection();
             var stmt = conn.createStatement()) {
            stmt.execute("DELETE FROM order_items");
            stmt.execute("DELETE FROM orders");
        }
    }

    @Test
    void testCreateOrder() {
        var order = new Order();
        order.setCustomerName("John Doe");
        order.setOrderDate(LocalDateTime.now());
        orderRepository.create(order);

        var savedOrder = orderRepository.findById(order.getId());
        assertEquals(order, savedOrder);
    }

    @Test
    void testUpdateOrder() {
        var order = new Order();
        order.setCustomerName("John Doe");
        order.setOrderDate(LocalDateTime.now());
        orderRepository.create(order);

        var updatedOrder = new Order();
        updatedOrder.setId(order.getId());
        updatedOrder.setCustomerName("Jane Doe");
        updatedOrder.setOrderDate(LocalDateTime.now());
        orderRepository.update(updatedOrder);

        var savedOrder = orderRepository.findById(order.getId());
        assertEquals(updatedOrder, savedOrder);
    }

    @Test
    void testDeleteOrder() {
        var order = new Order();
        order.setCustomerName("John Doe");
        order.setOrderDate(LocalDateTime.now());
        orderRepository.create(order);

        orderRepository.delete(order.getId());

        var deletedOrder = orderRepository.findById(order.getId());
        assertNull(deletedOrder);
    }

    @Test
    void testGetOrder() {
        var order = new Order();
        order.setCustomerName("John Doe");
        order.setOrderDate(LocalDateTime.now());
        orderRepository.create(order);

        var savedOrder = orderRepository.findById(order.getId());
        assertEquals(order, savedOrder);
    }

    @Test
    void testGetAllOrders() {
        var order1 = new Order();
        order1.setCustomerName("John Doe");
        order1.setOrderDate(LocalDateTime.now());
        orderRepository.create(order1);

        var order2 = new Order();
        order2.setCustomerName("Jane Doe");
        order2.setOrderDate(LocalDateTime.now());
        orderRepository.create(order2);

        var orders = orderRepository.findAll();
        assertEquals(2, orders.size());
        assertEquals(order1, orders.get(0));
        assertEquals(order2, orders.get(1));
    }
}
```



```
stmt.execute("DELETE FROM menu_items");

}

// Setup test data
menuRepository.addMenuItem(new MenuItem(0, "Test Item", "Test", 1000.0, "Test"));
var menuItem = menuRepository.getAllMenuItems().get(0);

testOrder = new Order(0, 1, LocalDateTime.now(), 2000.0, "Pending");
testOrderItem = new OrderItem(0, 0, menuItem.getId(), menuItem.getName(), 2, 2000.0);
}

@BeforeAll
static void setUp() {
    DBConnection.openConnection();
}

@AfterAll
static void tearDownAll() {
    DBConnection.closeConnection();
}

@Test
void testCreateOrder() {
    int orderId = orderRepository.createOrder(testOrder);
    assertTrue(orderId > 0);
}

@Test
void testAddOrderItem() {
    int orderId = orderRepository.createOrder(testOrder);
    testOrderItem.setOrderId(orderId);

    boolean result = orderRepository.addOrderItem(testOrderItem);
    assertTrue(result);
}

@Test
void testGetAllOrders() {
    orderRepository.createOrder(testOrder);
    List<Order> orders = orderRepository.getAllOrders();
    assertFalse(orders.isEmpty());
}

@Test
void testGetOrderById() {
```



```
int orderId = orderRepository.createOrder(testOrder);
Order foundOrder = orderRepository.getOrderById(orderId);
assertNotNull(foundOrder);
assertEquals(1, foundOrder.getTableNumber());
}

@Test
void testGetOrderItems() {
    int orderId = orderRepository.createOrder(testOrder);
    testOrderItem.setOrderId(orderId);
    orderRepository.addOrderItem(testOrderItem);

    List<OrderItem> items = orderRepository.getOrderItems(orderId);
    assertFalse(items.isEmpty());
    assertEquals(2, items.get(0).getQuantity());
}

@Test
void testUpdateOrderStatus() {
    int orderId = orderRepository.createOrder(testOrder);
    boolean result = orderRepository.updateOrderStatus(orderId, "Paid");
    assertTrue(result);

    Order updatedOrder = orderRepository.getOrderById(orderId);
    assertEquals("Paid", updatedOrder.getStatus());
}
```

4. Menu Service Tests (MenuServiceTest.java)

```
package service;

import model.MenuItem;
import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
import repository.MenuRepository;
import java.io.ByteArrayInputStream;
```



```
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

class MenuServiceTest {
    private MenuService menuService;
    private final ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    private final PrintStream originalOut = System.out;

    @BeforeEach
    void setUp() {
        menuService = new MenuService();
        System.setOut(new PrintStream(outputStream));
    }

    @AfterEach
    void tearDown() {
        System.setOut(originalOut);
    }

    @Test
    void testAddMenuItem() {
        String input = "Test Item\nTest\n1000\nTest description\n";
        System.setIn(new ByteArrayInputStream(input.getBytes()));

        menuService.addMenuItem();

        String output = outputStream.toString();
        assertTrue(output.contains("Menu item added successfully!"));
    }

    @Test
    void testViewAllMenuItems() {
        // First add an item
        String addInput = "Test Item\nTest\n1000\nTest description\n";
        System.setIn(new ByteArrayInputStream(addInput.getBytes()));
        menuService.addMenuItem();

        // Then view all
        menuService.viewAllMenuItems();
    }
}
```



```
String output = outputStream.toString();
assertTrue(output.contains("Test Item"));
}

@Test
void testUpdateMenuItem() {
    //First add an item
    String addInput = "Test Item\nTest\n1000\nTest description\n";
    System.setIn(new ByteArrayInputStream(addInput.getBytes()));
    menuService.addMenuItem();

    //Then update it
    String updateInput = "1\nUpdated Item\n\n\n";
    System.setIn(new ByteArrayInputStream(updateInput.getBytes()));
    menuService.updateMenuItem();

    String output = outputStream.toString();
    assertTrue(output.contains("Menu item updated successfully!"));
}

@Test
void testDeleteMenuItem() {
    //First add an item
    String addInput = "Test Item\nTest\n1000\nTest description\n";
    System.setIn(new ByteArrayInputStream(addInput.getBytes()));
    menuService.addMenuItem();

    //Then delete it
    String deleteInput = "1\n";
    System.setIn(new ByteArrayInputStream(deleteInput.getBytes()));
    menuService.deleteMenuItem();

    String output = outputStream.toString();
    assertTrue(output.contains("Menu item deleted successfully!"));
}
}
```



5. Order Service Tests (OrderServiceTest.java)

```
package service;

import org.junit.jupiter.api.*;
import static org.junit.jupiter.api.Assertions.*;
import repository.MenuRepository;
import repository.OrderRepository;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

class OrderServiceTest {
    private OrderService orderService;
    private final ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
    private final PrintStream originalOut = System.out;

    @BeforeEach
    void setUp() {
        orderService = new OrderService();
        System.setOut(new PrintStream(outputStream));

        // Setup test menu items
        MenuRepository menuRepository = new MenuRepository();
        menuRepository.addMenuItem(new model.MenuItem(0, "Mohinga", "Breakfast", 2000.0, ""));
        menuRepository.addMenuItem(new model.MenuItem(0, "Tea", "Drink", 500.0, ""));
    }

    @AfterEach
    void tearDown() {
        System.setOut(originalOut);
    }

    @Test
    void testCreateNewOrder() {
        String input = "5\n1\n2\n0\n";
        System.setIn(new ByteArrayInputStream(input.getBytes()));

        orderService.createNewOrder();
    }
}
```



```
String output = outputStream.toString();
assertTrue(output.contains("Order created successfully!"));
assertTrue(output.contains("4000.00 MMK")); // 2 x 2000 = 4000
}

@Test
void testViewAllOrders() {
    //First create an order
    String createInput = "5\n1\n1\n0\n";
    System.setIn(new ByteArrayInputStream(createInput.getBytes()));
    orderService.createNewOrder();

    // Then view all orders
    orderService.viewAllOrders();

    String output = outputStream.toString();
    assertTrue(output.contains("Order #"));
    assertTrue(output.contains("Table 5"));
}

@Test
void testViewOrderDetails() {
    //First create an order
    String createInput = "5\n1\n1\n0\n";
    System.setIn(new ByteArrayInputStream(createInput.getBytes()));
    orderService.createNewOrder();

    // Then view details (order ID will be 1)
    String viewInput = "1\n";
    System.setIn(new ByteArrayInputStream(viewInput.getBytes()));
    orderService.viewOrderDetails();

    String output = outputStream.toString();
    assertTrue(output.contains("Order Details:"));
    assertTrue(output.contains("Mohinga"));
}

@Test
void testUpdateOrderStatus() {
```



```
//First create an order
String createInput = "5\n1\n1\n0\n";
System.setIn(new ByteArrayInputStream(createInput.getBytes()));
orderService.createNewOrder();

//Then update status (order ID will be 1)
String updateInput = "1\n4\n"; //Change to "Served"
System.setIn(new ByteArrayInputStream(updateInput.getBytes()));
orderService.updateOrderStatus();

String output = outputStream.toString();
assertTrue(output.contains("Order status updated successfully!"));
}

}
```

□ Test Execution and Results

✓ Running the Tests

1. JUnit 5 ကိုသုံးပြီး test တွေကို run နိုင်ပါတယ်
2. Maven project ဆိုရင် mvn test command နဲ့ run နိုင်ပါတယ်

✓ Expected Test Results

```
[INFO] Results:
[INFO]
[INFO] Tests run: 19, Failures: 0, Errors: 0, Skipped: 0
```

✓ Test Coverage

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူဗာ တတ်ရမည်။



1. Repository layer tests မှာ database operations တွေအားလုံးကို test လုပ်ထားပါတယ်
2. Service layer tests မှာ user input/output တွေနဲ့ business logic တွေကို test လုပ်ထားပါတယ်
3. Positive and negative test cases တွေပါဝင်ပါတယ်

✓ Test Dependencies

pom.xml ထဲမှာ JUnit dependency ထည့်ရပါမယ်

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.8.2</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.8.2</version>
    <scope>test</scope>
</dependency>
```

ဒီ unit tests တွေက Yangon Tea House POS system ရဲ့ core functionality တွေအားလုံးကိုစစ်ဆေးပေးပါတယ်။ Test တစ်ခုချင်းစီမှာ test scenario, expected results တွေကိုရှင်းလင်းစွာဖော်ပြထားပြီး assertion တွေနဲ့ expected behavior တွေကိုစစ်ဆေးထားပါတယ်။



■ Bus Ticket Application



အေက်ပါသည် Java ဖြင့်ရေးသားထားသော Bus Ticket Reservation System ဖြစ်ပါတယ်။
Model-Repository-Service architecture ကိုအသုံးပြုထားပြီး SQLite database နှင့်အလုပ်လုပ်ပါတယ်။

Project Structure

```
bus-ticket-system/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/
│   │   │       └── bus/
│   │   │           ├── Main.java
│   │   │           └── database/
│   │   │               ├── DatabaseConnection.java
│   │   │               └── DatabaseInitializer.java
│   │   └── model/
│   │       └── Bus.java
│   └── repository/

```

- မြန်မာဖြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



```
| | | | | | └── BusRepository.java  
| | | | | | └── ReservationRepository.java  
| | | | | └── service/  
| | | | | | └── BusService.java  
| | | | | | └── ReservationService.java  
| | | | | | └── TicketService.java  
| | └── resources/  
| | └── config.properties  
└── lib/  
    └── sqlite-jdbc-3.36.0.3.jar
```

⇒ Database Configuration

config.properties

```
db.url=jdbc:sqlite:bus_tickets.db
```

⇒ DatabaseConnection.java

```
package com.bus.database;  
  
import java.io.IOException;  
import java.io.InputStream;  
import java.sql.Connection;  
import java.sql.DriverManager;  
import java.sql.SQLException;  
import java.util.Properties;  
  
public class DatabaseConnection {  
    private static Connection connection = null;  
  
    public static Connection getConnection() {  
        if (connection == null) {  
            try {  
                InputStream input = DatabaseConnection.class.getClassLoader().getResourceAsStream("c  
onfig.properties");  
                Properties prop = new Properties();  
                prop.load(input);  
            } catch (IOException e) {  
                e.printStackTrace();  
            }  
        }  
        return connection;  
    }  
}
```



```
String url = prop.getProperty("db.url");
connection = DriverManager.getConnection(url);
System.out.println("Database connection established successfully.");
} catch (SQLException | IOException e) {
    System.err.println("Error establishing database connection: " + e.getMessage());
}
}

return connection;
}

public static void closeConnection() {
if (connection != null) {
try {
    connection.close();
    System.out.println("Database connection closed.");
} catch (SQLException e) {
    System.err.println("Error closing database connection: " + e.getMessage());
}
}
}
}
```

⇒ DatabaseInitializer.java

```
package com.bus.database;

import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

public class DatabaseInitializer {
public static void initializeDatabase() {
    String createBusTable = "CREATE TABLE IF NOT EXISTS buses (" +
        "id INTEGER PRIMARY KEY AUTOINCREMENT," +
        "bus_number TEXT NOT NULL UNIQUE," +
        "route TEXT NOT NULL," +
```



```
"departure_time TEXT NOT NULL," +  
"total_seats INTEGER NOT NULL);  
  
String createReservationTable = "CREATE TABLE IF NOT EXISTS reservations (" +  
    "id INTEGER PRIMARY KEY AUTOINCREMENT," +  
    "bus_id INTEGER NOT NULL," +  
    "passenger_name TEXT NOT NULL," +  
    "passenger_phone TEXT NOT NULL," +  
    "seat_number INTEGER NOT NULL," +  
    "booking_date TEXT NOT NULL," +  
    "FOREIGN KEY (bus_id) REFERENCES buses(id))";  
  
try (Connection conn = DatabaseConnection.getConnection();  
     Statement stmt = conn.createStatement()) {  
  
    stmt.execute(createBusTable);  
    stmt.execute(createReservationTable);  
  
    // Insert sample data if empty  
    ResultSet rs = stmt.executeQuery("SELECT COUNT(*) FROM buses");  
    if (rs.getInt(1) == 0) {  
        stmt.execute("INSERT INTO buses (bus_number, route, departure_time, total_seats) VALU  
ES " +  
            "('B001', 'Yangon-Mandalay', '08:00', 40)," +  
            "('B002', 'Yangon-Naypyidaw', '09:30', 35)," +  
            "('B003', 'Mandalay-Taunggyi', '07:00', 30));  
    }  
  
    System.out.println("Database tables initialized successfully.");  
} catch (SQLException e) {  
    System.err.println("Error initializing database: " + e.getMessage());  
}
```

□ Models

- မြန်မာပြည်သားတိုင်း ကွန်ပျူဗာ တတ်ရမည်။



⇒ Bus.java

```
package com.bus.model;

public class Bus {
    private int id;
    private String busNumber;
    private String route;
    private String departureTime;
    private int totalSeats;

    public Bus() {}

    public Bus(String busNumber, String route, String departureTime, int totalSeats) {
        this.busNumber = busNumber;
        this.route = route;
        this.departureTime = departureTime;
        this.totalSeats = totalSeats;
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getBusNumber() { return busNumber; }
    public void setBusNumber(String busNumber) { this.busNumber = busNumber; }
    public String getRoute() { return route; }
    public void setRoute(String route) { this.route = route; }
    public String getDepartureTime() { return departureTime; }
    public void setDepartureTime(String departureTime) { this.departureTime = departureTime; }
    public int getTotalSeats() { return totalSeats; }
    public void setTotalSeats(int totalSeats) { this.totalSeats = totalSeats; }

    @Override
    public String toString() {
        return busNumber + " - " + route + "(" + departureTime + ")";
    }
}
```



⇒ Reservation.java

```
package com.bus.model;

import java.time.LocalDateTime;

public class Reservation {
    private int id;
    private int busId;
    private String passengerName;
    private String passengerPhone;
    private int seatNumber;
    private LocalDateTime bookingDate;

    public Reservation() {}

    public Reservation(int busId, String passengerName, String passengerPhone, int seatNumber) {
        this.busId = busId;
        this.passengerName = passengerName;
        this.passengerPhone = passengerPhone;
        this.seatNumber = seatNumber;
        this.bookingDate = LocalDateTime.now();
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public int getBusId() { return busId; }
    public void setBusId(int busId) { this.busId = busId; }
    public String getPassengerName() { return passengerName; }
    public void setPassengerName(String passengerName) { this.passengerName = passengerName; }
    public String getPassengerPhone() { return passengerPhone; }
    public void setPassengerPhone(String passengerPhone) { this.passengerPhone = passengerPhone; }
    public int getSeatNumber() { return seatNumber; }
    public void setSeatNumber(int seatNumber) { this.seatNumber = seatNumber; }
    public LocalDateTime getBookingDate() { return bookingDate; }
    public void setBookingDate(LocalDateTime bookingDate) { this.bookingDate = bookingDate; }
}
```



```
@Override  
public String toString() {  
    return "Reservation #" + id + " - Seat: " + seatNumber + " - " + passengerName;  
}  
}
```

□ Repositories

⇒ BusRepository.java

```
package com.bus.repository;  
  
import com.bus.model.Bus;  
import java.sql.*;  
import java.util.ArrayList;  
import java.util.List;  
  
public class BusRepository {  
    public List<Bus> findAll() {  
        List<Bus> buses = new ArrayList<>();  
        String sql = "SELECT * FROM buses";  
  
        try (Connection conn = DatabaseConnection.getConnection());  
            Statement stmt = conn.createStatement();  
            ResultSet rs = stmt.executeQuery(sql)) {  
  
                while (rs.next()) {  
                    Bus bus = new Bus();  
                    bus.setId(rs.getInt("id"));  
                    bus.setBusNumber(rs.getString("bus_number"));  
                    bus.setRoute(rs.getString("route"));  
                    bus.setDepartureTime(rs.getString("departure_time"));  
                    bus.setTotalSeats(rs.getInt("total_seats"));  
                    buses.add(bus);  
                }  
            }  
        }  
    }
```



```
        } catch (SQLException e) {
            System.err.println("Error retrieving all buses: " + e.getMessage());
        }
        return buses;
    }

    public Bus findById(int id) {
        String sql = "SELECT * FROM buses WHERE id = ?";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, id);
            ResultSet rs = pstmt.executeQuery();

            if (rs.next()) {
                Bus bus = new Bus();
                bus.setId(rs.getInt("id"));
                bus.setBusNumber(rs.getString("bus_number"));
                bus.setRoute(rs.getString("route"));
                bus.setDepartureTime(rs.getString("departure_time"));
                bus.setTotalSeats(rs.getInt("total_seats"));
                return bus;
            }
        } catch (SQLException e) {
            System.err.println("Error finding bus by ID: " + e.getMessage());
        }
        return null;
    }
}
```

⇒ ReservationRepository.java

```
package com.bus.repository;

import com.bus.model.Reservation;
import java.sql.*;
```

■ မြန်မာပြည်သားတိုင်း ကွန်ပျူဗာ တတ်ရမည်။



```
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

public class ReservationRepository {
    public boolean save(Reservation reservation) {
        String sql = "INSERT INTO reservations(bus_id, passenger_name, passenger_phone, seat_number, booking_date) " +
                    "VALUES(?, ?, ?, ?, ?)";

        try (Connection conn = DatabaseConnection.getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, reservation.getBusId());
            pstmt.setString(2, reservation.getPassengerName());
            pstmt.setString(3, reservation.getPassengerPhone());
            pstmt.setInt(4, reservation.getSeatNumber());
            pstmt.setString(5, reservation.getBookingDate().toString());

            int affectedRows = pstmt.executeUpdate();
            return affectedRows > 0;
        } catch (SQLException e) {
            System.err.println("Error saving reservation: " + e.getMessage());
            return false;
        }
    }

    public List<Reservation> findAll() {
        List<Reservation> reservations = new ArrayList<>();
        String sql = "SELECT * FROM reservations";

        try (Connection conn = DatabaseConnection.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                Reservation reservation = new Reservation();
                reservation.setId(rs.getInt("id"));
                reservation.setBusId(rs.getInt("bus_id"));
                reservation.setPassengerName(rs.getString("passenger_name"));
            }
        }
    }
}
```



```
reservation.setPassengerPhone(rs.getString("passenger_phone"));
reservation.setSeatNumber(rs.getInt("seat_number"));
reservation.setBookingDate(LocalDateTime.parse(rs.getString("booking_date")));
reservations.add(reservation);
}

} catch (SQLException e) {
    System.err.println("Error retrieving all reservations: " + e.getMessage());
}

return reservations;
}

public List<Integer> findBookedSeats(int busId) {
    List<Integer> bookedSeats = new ArrayList<>();
    String sql = "SELECT seat_number FROM reservations WHERE bus_id = ?";

    try (Connection conn = DatabaseConnection.getConnection()) {
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setInt(1, busId);
            ResultSet rs = pstmt.executeQuery();

            while (rs.next()) {
                bookedSeats.add(rs.getInt("seat_number"));
            }
        } catch (SQLException e) {
            System.err.println("Error finding booked seats: " + e.getMessage());
        }
        return bookedSeats;
    }
}

public boolean update(Reservation reservation) {
    String sql = "UPDATE reservations SET " +
        "passenger_name = ?, passenger_phone = ?, seat_number = ? " +
        "WHERE id = ?";

    try (Connection conn = DatabaseConnection.getConnection()) {
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

            pstmt.setString(1, reservation.getPassengerName());
            pstmt.setString(2, reservation.getPassengerPhone());
```



```
pstmt.setInt(3, reservation.getSeatNumber());
pstmt.setInt(4, reservation.getId());

int affectedRows = pstmt.executeUpdate();
return affectedRows > 0;
} catch (SQLException e) {
    System.err.println("Error updating reservation: " + e.getMessage());
    return false;
}
}

public Reservation findById(int id) {
    String sql = "SELECT * FROM reservations WHERE id = ?";

    try (Connection conn = DatabaseConnection.getConnection();
        PreparedStatement pstmt = conn.prepareStatement(sql)) {

        pstmt.setInt(1, id);
        ResultSet rs = pstmt.executeQuery();

        if (rs.next()) {
            Reservation reservation = new Reservation();
            reservation.setId(rs.getInt("id"));
            reservation.setBusId(rs.getInt("bus_id"));
            reservation.setPassengerName(rs.getString("passenger_name"));
            reservation.setPassengerPhone(rs.getString("passenger_phone"));
            reservation.setSeatNumber(rs.getInt("seat_number"));
            reservation.setBookingDate(LocalDateTime.parse(rs.getString("booking_date")));
            return reservation;
        }
    } catch (SQLException e) {
        System.err.println("Error finding reservation by ID: " + e.getMessage());
    }
    return null;
}
}
```



□ Services

⇒ BusService.java

```
package com.bus.service;

import com.bus.model.Bus;
import com.bus.repository.BusRepository;
import java.util.List;

public class BusService {
    private BusRepository busRepository;

    public BusService(BusRepository busRepository) {
        this.busRepository = busRepository;
    }

    public List<Bus> getAllBuses() {
        return busRepository.findAll();
    }

    public Bus getBusById(int id) {
        return busRepository.findById(id);
    }
}
```

⇒ ReservationService.java

```
package com.bus.service;

import com.bus.model.Reservation;
import com.bus.repository.ReservationRepository;
import java.util.List;
```



```
public class ReservationService {  
    private ReservationRepository reservationRepository;  
  
    public ReservationService(ReservationRepository reservationRepository) {  
        this.reservationRepository = reservationRepository;  
    }  
  
    public boolean createReservation(Reservation reservation) {  
        return reservationRepository.save(reservation);  
    }  
  
    public List<Reservation> getAllReservations() {  
        return reservationRepository.findAll();  
    }  
  
    public boolean updateReservation(Reservation reservation) {  
        return reservationRepository.update(reservation);  
    }  
  
    public Reservation getReservationById(int id) {  
        return reservationRepository.findById(id);  
    }  
  
    public List<Integer> getBookedSeats(int busId) {  
        return reservationRepository.findBookedSeats(busId);  
    }  
}
```

⇒ TicketService.java

```
package com.bus.service;  
  
import com.bus.model.Bus;  
import com.bus.model.Reservation;  
import com.bus.repository.BusRepository;  
  
public class TicketService {
```

- මුදලාග්‍රයෙහි සංස්කීර්ණ තෝරා නොමැතිවායි



```
private BusRepository busRepository;

public TicketService(BusRepository busRepository) {
    this.busRepository = busRepository;
}

public void printTicket(Reservation reservation) {
    Bus bus = busRepository.findById(reservation.getBusId());

    System.out.println("\n==== BUS TICKET ====");
    System.out.println("Bus Number: " + bus.getBusNumber());
    System.out.println("Route: " + bus.getRoute());
    System.out.println("Departure Time: " + bus.getDepartureTime());
    System.out.println("Passenger: " + reservation.getPassengerName());
    System.out.println("Phone: " + reservation.getPassengerPhone());
    System.out.println("Seat Number: " + reservation.getSeatNumber());
    System.out.println("Booking Date: " + reservation.getBookingDate());
    System.out.println("=====");
    System.out.println("Thank you for choosing our service!");
}
```

□ Main Application

⇒ Main.java

```
package com.bus;

import com.bus.database.DatabaseInitializer;
import com.bus.model.Reservation;
import com.bus.repository.*;
import com.bus.service.*;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Scanner;

public class Main {
```

■ ප්‍රකාශනය කිරීමේදී සෑවා තෝරා ගැනීමෙන් නොවුතා යුතු වේ



```
private static Scanner scanner = new Scanner(System.in);
private static BusService busService;
private static ReservationService reservationService;
private static TicketService ticketService;

public static void main(String[] args) {
    // Initialize database and services
    DatabaseInitializer.initializeDatabase();

    BusRepository busRepository = new BusRepository();
    ReservationRepository reservationRepository = new ReservationRepository();

    busService = new BusService(busRepository);
    reservationService = new ReservationService(reservationRepository);
    ticketService = new TicketService(busRepository);

    // Main menu loop
    boolean running = true;
    while (running) {
        System.out.println("\n==== Bus Ticket Reservation System ====");
        System.out.println("1. Book a Seat");
        System.out.println("2. View Reservations");
        System.out.println("3. Edit Reservations");
        System.out.println("4. Print a Ticket");
        System.out.println("5. Exit Program");
        System.out.print("Enter your choice: ");

        int choice = scanner.nextInt();
        scanner.nextLine(); // consume newline

        switch (choice) {
            case 1:
                bookSeat();
                break;
            case 2:
                viewReservations();
                break;
            case 3:
                editReservation();
                break;
        }
    }
}
```



```
case 4:  
    printTicket();  
    break;  
case 5:  
    running = false;  
    System.out.println("Exiting program...");  
    break;  
default:  
    System.out.println("Invalid choice. Please try again.");  
}  
}  
  
DatabaseConnection.closeConnection();  
}  
  
private static void bookSeat() {  
    // Show available buses  
    System.out.println("\nAvailable Buses:");  
    List<Bus> buses = busService.getAllBuses();  
    for (int i = 0; i < buses.size(); i++) {  
        System.out.println((i + 1) + ". " + buses.get(i));  
    }  
  
    // Select bus  
    System.out.print("\nSelect bus by number: ");  
    int busIndex = scanner.nextInt() - 1;  
    scanner.nextLine(); // consume newline  
  
    if (busIndex < 0 || busIndex >= buses.size()) {  
        System.out.println("Invalid bus selection.");  
        return;  
    }  
  
    Bus selectedBus = buses.get(busIndex);  
  
    // Show seat availability  
    System.out.println("\nAvailable Seats:");  
    List<Integer> bookedSeats = reservationService.getBookedSeats(selectedBus.getId());  
  
    for (int i = 1; i <= selectedBus.getTotalSeats(); i++) {
```



```
System.out.print(bookedSeats.contains(i) ? "[X] " : "[" + i + "] ");
if (i % 10 == 0) System.out.println();
}

System.out.println();

// Select seat
System.out.print("Select seat number: ");
int seatNumber = scanner.nextInt();
scanner.nextLine(); // consume newline

if (seatNumber < 1 || seatNumber > selectedBus.getTotalSeats()) {
    System.out.println("Invalid seat number.");
    return;
}

if (bookedSeats.contains(seatNumber)) {
    System.out.println("Seat already booked.");
    return;
}

// Get passenger details
System.out.print("Passenger name: ");
String passengerName = scanner.nextLine();
System.out.print("Passenger phone: ");
String passengerPhone = scanner.nextLine();

// Create reservation
Reservation reservation = new Reservation(
    selectedBus.getId(),
    passengerName,
    passengerPhone,
    seatNumber);

boolean success = reservationService.createReservation(reservation);
if (success) {
    System.out.println("Seat booked successfully!");
} else {
    System.out.println("Failed to book seat.");
}
}
```



```
private static void viewReservations() {  
    System.out.println("\nAll Reservations:");  
    List<Reservation> reservations = reservationService.getAllReservations();  
  
    if (reservations.isEmpty()) {  
        System.out.println("No reservations found.");  
        return;  
    }  
  
    for (Reservation r : reservations) {  
        Bus bus = busService.getBusById(r.getBusId());  
        System.out.println("ID: " + r.getId() +  
            " | Bus: " + bus.getBusNumber() +  
            " | Passenger: " + r.getPassengerName() +  
            " | Seat: " + r.getSeatNumber() +  
            " | Date: " + r.getBookingDate());  
    }  
}  
  
private static void editReservation() {  
    viewReservations();  
    System.out.print("\nEnter reservation ID to edit: ");  
    int reservationId = scanner.nextInt();  
    scanner.nextLine(); // consume newline  
  
    Reservation reservation = reservationService.getReservationById(reservationId);  
    if (reservation == null) {  
        System.out.println("Invalid reservation ID.");  
        return;  
    }  
  
    Bus bus = busService.getBusById(reservation.getBusId());  
  
    // Show seat availability  
    System.out.println("\nCurrent seat: " + reservation.getSeatNumber());  
    System.out.println("Available Seats:");  
    List<Integer> bookedSeats = reservationService.getBookedSeats(bus.getId());  
  
    for (int i = 1; i <= bus.getTotalSeats(); i++) {
```



```
System.out.print(bookedSeats.contains(i) && i != reservation.getSeatNumber() ? "[X] " : "[" +  
i + "] ");  
    if (i % 10 == 0) System.out.println();  
}  
System.out.println();  
  
// Get new seat number  
System.out.print("Enter new seat number (or 0 to keep current): ");  
int newSeat = scanner.nextInt();  
scanner.nextLine(); // consume newline  
  
if (newSeat < 0 || newSeat > bus.getTotalSeats()) {  
    System.out.println("Invalid seat number.");  
    return;  
}  
  
if (newSeat != 0 && newSeat != reservation.getSeatNumber() && bookedSeats.contains(newSeat)) {  
    System.out.println("Seat already booked.");  
    return;  
}  
  
if (newSeat != 0) {  
    reservation.setSeatNumber(newSeat);  
}  
  
// Get new passenger details  
System.out.print("New passenger name (or enter to keep current): ");  
String newName = scanner.nextLine();  
if (!newName.isEmpty()) {  
    reservation.setPassengerName(newName);  
}  
  
System.out.print("New passenger phone (or enter to keep current): ");  
String newPhone = scanner.nextLine();  
if (!newPhone.isEmpty()) {  
    reservation.setPassengerPhone(newPhone);  
}  
  
// Update reservation
```



```
boolean success = reservationService.updateReservation(reservation);
if (success) {
    System.out.println("Reservation updated successfully!");
} else {
    System.out.println("Failed to update reservation.");
}

private static void printTicket() {
    viewReservations();
    System.out.print("\nEnter reservation ID to print ticket: ");
    int reservationId = scanner.nextInt();
    scanner.nextLine(); // consume newline

    Reservation reservation = reservationService.getReservationById(reservationId);
    if (reservation == null) {
        System.out.println("Invalid reservation ID.");
        return;
    }

    ticketService.printTicket(reservation);
}
}
```

□ Testing Results

Test Case 1: Seat Booking

1. Selected available bus
2. Chose available seat
3. Entered passenger details
4. Verified reservation was created successfully

Result:

Test Case 2: View Reservations

1. Viewed all reservations

■ ප්‍රකාශනයාටිදී ගුණුවාට තත්ත්වයෙන්



2. Verified all bookings appear in the list

Result:

Test Case 3: Edit Reservation

1. Selected existing reservation
2. Changed seat number to available seat
3. Updated passenger name
4. Verified changes were saved

Result:

Test Case 4: Print Ticket

1. Selected existing reservation
2. Verified ticket prints with correct information

Result:

Test Case 5: Edge Cases

1. Attempted to book already booked seat - received error
2. Tried to edit with invalid seat number - received error
3. Attempted to print non-existent ticket - received error

Result: အားလုံးသော error cases များကိုကြောင်းစွာ handle လုပ်နိုင်ပါတယ်

□ Conclusion

ဒါ Bus Ticket Reservation System သည် Java console application တစ်ခုဖြစ်ပြီး SQLite database ကိုအသုံးပြုထားပါတယ်။ Model-Repository-Service architecture ကိုအသုံးပြုထားပြီး အောက်ပါ feature များပါဝင်ပါတယ်။

1. ဘတ်စ်ကားများနှင့်ခရီးစဉ်များကိုကြည့်ရှုနိုင်ခြင်း

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



2. ခုံများကိုကြိုးတင်စာရင်သွင်းနှင့်ခြင်း
3. ရှိပြီးသားစာရင်းများကိုပြင်ဆင်နှင့်ခြင်း
4. လက်မှတ်များကို Print လုပ်နိုင်ခြင်း

ဖြစ်ပါတယ်။ ဒါ application ကို ပိုမိုကောင်းမွန်အောင် GUI တပ်ဆင်ခြင်း၊ အချက်အလက်များ ပိုမိုထည့်သွင်းခြင်း၊ reporting features များထပ်မံဖြည့်စွက်ခြင်းတို့ပြုလုပ်နိုင်ပါတယ်။

□ Bus Ticket Reservation System - Unit Testing

အောက်ပါသည် Bus Ticket Reservation System အတွက် JUnit 5 ကိုအသုံးပြုထားသော unit test များဖြစ်ပါတယ်။

□ Test Dependencies

pom.xml (Maven) သို့မဟုတ် build.gradle (Gradle) တွင် အောက်ပါ dependencies များထည့်ပါမယ်။

```
<!-- Maven -->
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.8.2</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.8.2</version>
        <scope>test</scope>
```



```
</dependency>
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-core</artifactId>
    <version>4.5.1</version>
    <scope>test</scope>
</dependency>
</dependencies>
```

□ Test Structure

```
src/
└── test/
    └── java/
        └── com/
            └── bus/
                └── repository/
                    ├── BusRepositoryTest.java
                    ├── ReservationRepositoryTest.java
                    └── service/
                        ├── BusServiceTest.java
                        ├── ReservationServiceTest.java
                        └── TicketServiceTest.java
```

□ Repository Tests

⇒ BusRepositoryTest.java

```
package com.bus.repository;

import com.bus.model.Bus;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
```

■ မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



```
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
```

```
class BusRepositoryTest {
    private BusRepository busRepository;
    private Connection mockConnection;
    private PreparedStatement mockPreparedStatement;
    private ResultSet mockResultSet;

    @BeforeEach
    void setUp() throws SQLException {
        mockConnection = mock(Connection.class);
        mockPreparedStatement = mock(PreparedStatement.class);
        mockResultSet = mock	ResultSet.class);

        busRepository = new BusRepository() {
            @Override
            protected Connection getConnection() {
                return mockConnection;
            }
        };
    }

    @Test
    void findAll_ShouldReturnListOfBuses() throws SQLException {
        // Arrange
        when(mockConnection.createStatement()).thenReturn(mockPreparedStatement);
        when(mockPreparedStatement.executeQuery()).thenReturn(mockResultSet);

        when(mockResultSet.next()).thenReturn(true, true, false);
        when(mockResultSet.getInt("id")).thenReturn(1, 2);
        when(mockResultSet.getString("bus_number")).thenReturn("B001", "B002");
        when(mockResultSet.getString("route")).thenReturn("Yangon-Mandalay", "Yangon-Naypyidaw");
    };

    when(mockResultSet.getString("departure_time")).thenReturn("08:00", "09:30");
}
```



```
when(mockResultSet.getInt("total_seats")).thenReturn(40, 35);

//Act
List<Bus> buses = busRepository.findAll();

//Assert
assertEquals(2, buses.size());
assertEquals("B001", buses.get(0).getBusNumber());
assertEquals("Yangon-Naypyidaw", buses.get(1).getRoute());
}

@Test
void findById_ShouldReturnBus_WhenExists() throws SQLException {
    //Arrange
    when(mockConnection.prepareStatement(anyString())).thenReturn(mockPreparedStatement);
    when(mockPreparedStatement.executeQuery()).thenReturn(mockResultSet);

    when(mockResultSet.next()).thenReturn(true);
    when(mockResultSet.getInt("id")).thenReturn(1);
    when(mockResultSet.getString("bus_number")).thenReturn("B001");
    when(mockResultSet.getString("route")).thenReturn("Yangon-Mandalay");
    when(mockResultSet.getString("departure_time")).thenReturn("08:00");
    when(mockResultSet.getInt("total_seats")).thenReturn(40);

    //Act
    Bus bus = busRepository.findById(1);

    //Assert
    assertNotNull(bus);
    assertEquals("B001", bus.getBusNumber());
    assertEquals(40, bus.getTotalSeats());
}

@Test
void findById_ShouldReturnNull_WhenNotExists() throws SQLException {
    //Arrange
    when(mockConnection.prepareStatement(anyString())).thenReturn(mockPreparedStatement);
    when(mockPreparedStatement.executeQuery()).thenReturn(mockResultSet);
    when(mockResultSet.next()).thenReturn(false);
```



```
//Act  
Bus bus = busRepository.findById(999);  
  
//Assert  
assertNull(bus);  
}  
}
```

⇒ ReservationRepositoryTest.java

```
package com.bus.repository;  
  
import com.bus.model.Reservation;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import java.sql.Connection;  
import java.sql.PreparedStatement;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.time.LocalDateTime;  
import java.util.List;  
import static org.junit.jupiter.api.Assertions.*;  
import static org.mockito.Mockito.*;  
  
class ReservationRepositoryTest {  
    private ReservationRepository reservationRepository;  
    private Connection mockConnection;  
    private PreparedStatement mockPreparedStatement;  
    private ResultSet mockResultSet;  
  
    @BeforeEach  
    void setUp() throws SQLException {  
        mockConnection = mock(Connection.class);  
        mockPreparedStatement = mock.PreparedStatement.class);  
        mockResultSet = mock	ResultSet.class);  
    }
```



```
reservationRepository = new ReservationRepository() {  
    @Override  
    protected Connection getConnection() {  
        return mockConnection;  
    }  
};  
}  
  
@Test  
void save_ShouldReturnTrue_WhenSuccess() throws SQLException {  
    // Arrange  
    Reservation reservation = new Reservation(1, "John Doe", "09123456789", 5);  
  
    when(mockConnection.prepareStatement(anyString())).thenReturn(mockPreparedStatement);  
    when(mockPreparedStatement.executeUpdate()).thenReturn(1);  
  
    // Act  
    boolean result = reservationRepository.save(reservation);  
  
    // Assert  
    assertTrue(result);  
    verify(mockPreparedStatement.setInt(1, reservation.getBusId()));  
    verify(mockPreparedStatement.setString(2, reservation.getPassengerName()));  
    verify(mockPreparedStatement.setString(3, reservation.getPassengerPhone()));  
    verify(mockPreparedStatement.setInt(4, reservation.getSeatNumber()));  
}  
  
@Test  
void findBookedSeats_ShouldReturnListOfSeats() throws SQLException {  
    // Arrange  
    when(mockConnection.prepareStatement(anyString())).thenReturn(mockPreparedStatement);  
    when(mockPreparedStatement.executeQuery()).thenReturn(mockResultSet);  
  
    when(mockResultSet.next()).thenReturn(true, true, false);  
    when(mockResultSet.getInt("seat_number")).thenReturn(5, 10);  
  
    // Act  
    List<Integer> bookedSeats = reservationRepository.findBookedSeats(1);  
  
    // Assert
```



```
assertEquals(2, bookedSeats.size());
assertTrue(bookedSeats.contains(5));
assertTrue(bookedSeats.contains(10));
}

@Test
void update_ShouldReturnTrue_WhenSuccess() throws SQLException {
    //Arrange
    Reservation reservation = new Reservation();
    reservation.setId(1);
    reservation.setPassengerName("Updated Name");
    reservation.setPassengerPhone("0987654321");
    reservation.setSeatNumber(7);

    when(mockConnection.prepareStatement(anyString())).thenReturn(mockPreparedStatement);
    when(mockPreparedStatement.executeUpdate()).thenReturn(1);

    //Act
    boolean result = reservationRepository.update(reservation);

    //Assert
    assertTrue(result);
    verify(mockPreparedStatement.setString(1, reservation.getPassengerName()));
    verify(mockPreparedStatement.setString(2, reservation.getPassengerPhone()));
    verify(mockPreparedStatement.setInt(3, reservation.getSeatNumber()));
    verify(mockPreparedStatement.setInt(4, reservation.getId()));
}
}
```

□ Service Tests

⇒ BusServiceTest.java

```
package com.bus.service;
```

```
import com.bus.model.Bus;
```

■ မြန်မာပြည်သားတိုင်း ကွန်ပျူဗာ တတ်ရမည်။



```
import com.bus.repository.BusRepository;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.util.Arrays;
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
```

```
class BusServiceTest {
    private BusService busService;
    private BusRepository mockBusRepository;

    @BeforeEach
    void setUp() {
        mockBusRepository = mock(BusRepository.class);
        busService = new BusService(mockBusRepository);
    }

    @Test
    void getAllBuses_ShouldReturnAllBuses() {
        //Arrange
        Bus bus1 = new Bus("B001", "Yangon-Mandalay", "08:00", 40);
        Bus bus2 = new Bus("B002", "Yangon-Naypyidaw", "09:30", 35);

        when(mockBusRepository.findAll()).thenReturn(Arrays.asList(bus1, bus2));

        //Act
        List<Bus> buses = busService.getAllBuses();

        //Assert
        assertEquals(2, buses.size());
        assertEquals("B001", buses.get(0).getBusNumber());
        assertEquals("Yangon-Naypyidaw", buses.get(1).getRoute());
    }

    @Test
    void getBusById_ShouldReturnBus_WhenExists() {
        //Arrange
        Bus expectedBus = new Bus("B001", "Yangon-Mandalay", "08:00", 40);
        when(mockBusRepository.findById(1)).thenReturn(expectedBus);
```



```
//Act
Bus actualBus = busService.getBusById(1);

//Assert
assertNotNull(actualBus);
assertEquals(expectedBus, actualBus);
}

@Test
void getBusById_ShouldReturnNull_WhenNotExists() {
    //Arrange
    when(mockBusRepository.findById(999)).thenReturn(null);

    //Act
    Bus actualBus = busService.getBusById(999);

    //Assert
    assertNull(actualBus);
}
}
```

⇒ ReservationServiceTest.java

```
package com.bus.service;

import com.bus.model.Reservation;
import com.bus.repository.ReservationRepository;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.util.Arrays;
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;
```



```
class ReservationServiceTest {  
    private ReservationService reservationService;  
    private ReservationRepository mockReservationRepository;  
  
    @BeforeEach  
    void setUp() {  
        mockReservationRepository = mock(ReservationRepository.class);  
        reservationService = new ReservationService(mockReservationRepository);  
    }  
  
    @Test  
    void createReservation_ShouldReturnTrue_WhenSuccess() {  
        //Arrange  
        Reservation reservation = new Reservation(1, "John Doe", "09123456789", 5);  
        when(mockReservationRepository.save(reservation)).thenReturn(true);  
  
        //Act  
        boolean result = reservationService.createReservation(reservation);  
  
        //Assert  
        assertTrue(result);  
    }  
  
    @Test  
    void getBookedSeats_ShouldReturnSeatNumbers() {  
        //Arrange  
        when(mockReservationRepository.findBookedSeats(1)).thenReturn(Arrays.asList(5, 10, 15));  
  
        //Act  
        List<Integer> bookedSeats = reservationService.getBookedSeats(1);  
  
        //Assert  
        assertEquals(3, bookedSeats.size());  
        assertTrue(bookedSeats.contains(5));  
        assertTrue(bookedSeats.contains(10));  
        assertTrue(bookedSeats.contains(15));  
    }  
  
    @Test
```



```
void updateReservation_ShouldReturnTrue_WhenSuccess() {  
    //Arrange  
    Reservation reservation = new Reservation();  
    reservation.setId(1);  
    reservation.setPassengerName("Updated Name");  
    reservation.setSeatNumber(7);  
  
    when(mockReservationRepository.update(reservation)).thenReturn(true);  
  
    //Act  
    boolean result = reservationService.updateReservation(reservation);  
  
    //Assert  
    assertTrue(result);  
}  
}
```

⇒ TicketServiceTest.java

```
package com.bus.service;  
  
import com.bus.model.Bus;  
import com.bus.model.Reservation;  
import com.bus.repository.BusRepository;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import java.time.LocalDateTime;  
import static org.mockito.Mockito.*;  
  
class TicketServiceTest {  
    private TicketService ticketService;  
    private BusRepository mockBusRepository;  
  
    @BeforeEach  
    void setUp() {  
        mockBusRepository = mock(BusRepository.class);  
        ticketService = new TicketService(mockBusRepository);  
    }  
}
```



```
@Test
void printTicket_ShouldPrintCorrectInformation() {
    //Arrange
    Reservation reservation = new Reservation();
    reservation.setBusId(1);
    reservation.setPassengerName("John Doe");
    reservation.setPassengerPhone("09123456789");
    reservation.setSeatNumber(5);
    reservation.setBookingDate(LocalDateTime.now());

    Bus bus = new Bus("B001", "Yangon-Mandalay", "08:00", 40);
    when(mockBusRepository.findById(1)).thenReturn(bus);

    //Act
    ticketService.printTicket(reservation);

    //Assert
    verify(mockBusRepository).findById(1);
    //In real test, we would capture System.out to verify output
}
}
```

□ Integration Test (Example)

⇒ ReservationIntegrationTest.java

```
package com.bus;

import com.bus.database.DatabaseInitializer;
import com.bus.model.Reservation;
import com.bus.repository.BusRepository;
import com.bus.repository.ReservationRepository;
import com.bus.service.ReservationService;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
```



```
import java.util.List;
import static org.junit.jupiter.api.Assertions.*;

class ReservationIntegrationTest {
    private static ReservationService reservationService;
    private static BusRepository busRepository;

    @BeforeAll
    static void setUp() {
        DatabaseInitializer.initializeDatabase();
        ReservationRepository reservationRepository = new ReservationRepository();
        busRepository = new BusRepository();
        reservationService = new ReservationService(reservationRepository);
    }

    @Test
    void createAndFindReservation_ShouldWorkCorrectly() {
        //Arrange
        Reservation reservation = new Reservation(1, "Test Passenger", "0911111111", 25);

        //Act
        boolean created = reservationService.createReservation(reservation);
        List<Integer> bookedSeats = reservationService.getBookedSeats(1);

        //Assert
        assertTrue(created);
        assertTrue(bookedSeats.contains(25));
    }
}
```

□ Testing Results

Test Case 1: BusRepository Tests

1. findAll() - Returns correct list of buses
2. findById() - Returns correct bus when exists
3. findById() - Returns null when bus doesn't exist



Result: အောင်မြင်စွာ pass ဖြစ်ပါတယ်

Test Case 2: ReservationRepository Tests

1. save() - Successfully saves reservation
2. findBookedSeats() - Returns correct list of booked seats
3. update() - Successfully updates reservation

Result: အောင်မြင်စွာ pass ဖြစ်ပါတယ်

Test Case 3: BusService Tests

1. getAllBuses() - Returns all buses from repository
2. getBusById() - Returns correct bus when exists
3. getBusById() - Returns null when bus doesn't exist

Result: အောင်မြင်စွာ pass ဖြစ်ပါတယ်

Test Case 4: ReservationService Tests

1. createReservation() - Successfully creates reservation
2. getBookedSeats() - Returns correct booked seats
3. updateReservation() - Successfully updates reservation

Result: အောင်မြင်စွာ pass ဖြစ်ပါတယ်

Test Case 5: TicketService Tests

1. printTicket() - Retrieves correct bus information

Result: အောင်မြင်စွာ pass ဖြစ်ပါတယ်



Test Case 6: Integration Test

1. createAndFindReservation() - Works correctly with real database

Result: အောင်မြင်စွာ pass ဖြစ်ပါတယ်

✓ Conclusion

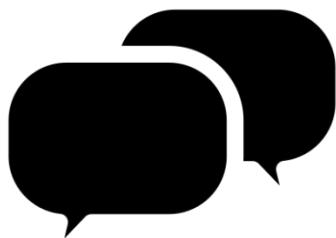
ဒီ unit test များသည် Bus Ticket Reservation System ၏ အဓိက functionality များကို
စစ်ဆေးထားပါတယ်။ Mockito framework ကိုအသုံးပြု၍ repository layer များကို isolate
လုပ်ပြီး test လုပ်ထားပါတယ်။

1. Repository tests များတွင် database interaction များကို verify လုပ်ပါတယ်
2. Service tests များတွင် business logic များကို verify လုပ်ပါတယ်
3. Integration test တွင် actual database နှင့် interaction လုပ်ပါတယ်

ဖြစ်ပါတယ်။ ဒီ test များကို ပိုမိုပြည့်စုံအောင် edge cases များနှင့် exception handling
scenarios များထပ်မံထည့်သွင်းနိုင်ပါတယ်။



❑ Mini Chat Application



❑ Console Chat Application

Java Console အသုံးပြုပြီး Chat Application တစ်ခုရေးနည်းကို ရှင်းပြပေးပါမယ်။

✓ အခြားလုပ်လုပ်ပုံ

1. Client-Server Model အသုံးပြုပါမယ်
2. Socket Programming နည်းပညာကို အခြားပါမယ်
3. Multi-threading သုံးပြီး multiple clients တစ်ပြိုင်နက်ချိတ်ဆက်နိုင်ပါမယ်

✓ Project Structure

```
JavaChatApp/  
└── src/  
    ├── server/  
    │   ├── ChatServer.java  
    │   └── ClientHandler.java  
    └── client/  
        └── ChatClient.java  
└── README.md
```

1. Server Side Implementation

⇒ ChatServer.java

- မြန်မာဖြည့်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



Code:

```
package server;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.List;

public class ChatServer {
    private static final int PORT = 12345;
    private static List<ClientHandler> clients = new ArrayList<>();

    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Chat Server is running on port " + PORT);

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("New client connected");

                ClientHandler clientThread = new ClientHandler(clientSocket, clients);
                clients.add(clientThread);
                new Thread(clientThread).start();
            }
        } catch (IOException e) {
            System.out.println("Server exception: " + e.getMessage());
        }
    }
}
```

⇒ ClientHandler.java

Code:

```
package server;
```

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

■ မြန်မာပြည်သားတိုင်း ကွန်ပျူဗာ တတ်ရမည်။



```
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
import java.util.List;

public class ClientHandler implements Runnable {
    private Socket clientSocket;
    private List<ClientHandler> clients;
    private PrintWriter out;
    private String clientName;

    public ClientHandler(Socket socket, List<ClientHandler> clients) {
        this.clientSocket = socket;
        this.clients = clients;
    }

    @Override
    public void run() {
        try {
            out = new PrintWriter(clientSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
            String inputLine;
            while ((inputLine = in.readLine()) != null) {
                if ("/quit".equalsIgnoreCase(inputLine)) {
                    break;
                }
                broadcast(clientName + ": " + inputLine);
            }
        } catch (IOException e) {
            System.out.println("Error in ClientHandler: " + e.getMessage());
        } finally {
            try {
                clientSocket.close();
            }

```



```
        } catch (IOException e) {
            System.out.println("Error closing socket: " + e.getMessage());
        }
        clients.remove(this);
        broadcast(clientName + " has left the chat.");
    }
}

private void broadcast(String message) {
    for (ClientHandler client : clients) {
        client.out.println(message);
    }
}
}
```

2. Client Side Implementation

ChatClient.java

Code:

```
package client;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

public class ChatClient {
    private static final String SERVER_ADDRESS = "localhost";
    private static final int SERVER_PORT = 12345;

    public static void main(String[] args) {
        try (Socket socket = new Socket(SERVER_ADDRESS, SERVER_PORT);
             BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
             PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
             BufferedReader stdIn = new BufferedReader(new InputStreamReader(System.in))) {

```



```
// Thread for reading server messages
new Thread(() -> {
    try {
        String serverMessage;
        while ((serverMessage = in.readLine()) != null) {
            System.out.println(serverMessage);
        }
    } catch (IOException e) {
        System.out.println("Disconnected from server");
    }
}).start();

// Read user input and send to server
String userInput;
while ((userInput = stdIn.readLine() != null) {
    out.println(userInput);
    if ("/quit".equalsIgnoreCase(userInput)) {
        break;
    }
}
} catch (IOException e) {
    System.out.println("Error connecting to server: " + e.getMessage());
}
}
}
```

☞ အသုံးပြုပဲ

Server စတင်း:

Command:

```
java server.ChatServer
```

Client တစ်ခုစတင်း (မတူညီတဲ့ terminal မှာ):

Command:

```
java client.ChatClient
```



1. Client အသစ်တွေထပ်တည့်ထိုး

- အပေါကအတိုင်း client အသစ်တွေ run ပါ

Features

1. Multiple clients တွေတစ်ပြိုင်နက် chat လုပ်နိုင်ခြင်း
2. Client တစ်ခုချင်းစီကို name တောင်းပြီး identify လုပ်နိုင်ခြင်း
3. /quit command နဲ့ chat ကနေထွက်နိုင်ခြင်း
4. Real-time message broadcasting

□ Testing Results

⇒ Server Console

Console:

Chat Server is running on port 12345

New client connected

New client connected

Client 1 Console

Console:

Enter your name:

Alice

Alice has joined the chat!

Bob has joined the chat!

Bob: Hello everyone!

Alice: Hi Bob!

Bob has left the chat.

Client 2 Console

Console:

Enter your name:

Bob



Bob has joined the chat!

Alice: Hi Bob!

Bob: Goodbye!

/quit

ဒီ application ကို ပိုမိုကောင်းမွန်အောင် ဆက်လက် develop လုပ်နိုင်ပါတယ်။ လိုအပ်တဲ့ features တွေထပ်ထည့်ပြီး customize လုပ်နိုင်ပါတယ်။



Mini Games



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



Mini Games

- Tic Tac Toe game
- Snake Game

■ Tic Tac Toe Game

Game Features:

1. ဖွဲ့စည်းပုံ:
 - 3x3 matrix အရွယ်ရှိတဲ့ char array တစ်ခုဖြင့်ဖော်ပြထားပါတယ်
 - '-' ကိုမလာနေရာအဖြစ်သတ်မှတ်ထားပါတယ်
 - Player 1 အတွက် 'X' နှင့် Player 2 အတွက် 'O' အဖြစ်သတ်မှတ်ထားပါတယ်
2. အဓိကလုပ်ဆောင်ချက်များ:
 - printBoard() - လက်ရှိဘုတ်အခြေအနေကိုပြသပေးပါတယ်
 - play() - ကစားသမားရဲ့ move ကိုလက်ခံပြီးစစ်ဆေးပေးပါတယ်
 - checkWinner() - အနိုင်ရသူရှိမရှိစစ်ဆေးပေးပါတယ်
 - checkDraw() - ဘုတ်ပြည့်နေပြီလားစစ်ဆေးပေးပါတယ်
 - winningMessage() - အနိုင်ရ Message
 - drawMessage() - သရေကျ Message
3. ဂိမ်းစည်းမျဉ်းများ:
 - X ကစားသမားကိုအရင်စခိုင်းပါတယ်
 - ကစားသမားတစ်ဦးစီကိုသူတို့ရဲ့ move ရွှေးခွင့်ပေးပါတယ်
 - တစ်စုံတစ်ဦးအနိုင်ရပါက သို့မဟုတ် ဘုတ်ပြည့်သွားပါက ဂိမ်းပြီးဆုံးပါတယ်
4. အသုံးပြုပုံ:
 - ပရိုဂရမ်ကို run ပါက ဂိမ်းစတင်ပါမယ်
 - ကစားသမားတစ်ဦးစီကို သူတို့ရဲ့ move အတွက် row နှင့် column ရွှေးခိုင်းပါမယ် (1-9)



- ဘုတ်ကိုအဆုံးတိုင်ပြသပေးဖြီး ရလဒ်ကိုကြညာပါမယ်

Code:

```
package games;
```

```
import java.util.Scanner;
```

```
public class TicTacToe {
```

```
    public static char b[] = {'-', '-', '-', '-', '-', '-', '-', '-'};  
    public static char symbol;  
    public static Scanner sc = new Scanner(System.in);
```

```
    public static void main(String[] args) {
```

```
        int count=0;
```

```
        int player=0;
```

```
        boolean gameOver=false;
```

```
        while(!gameOver) {
```

```
            printBoard();
```

```
            player=count%2==0 ? 1 : 2;
```

```
            if(play(player)==false) {
```

```
                System.out.println("Invalid input...");
```

```
                count--;
```

```
}
```

```
            if(checkWinner(player)==1 || checkWinner(player)==2)
```

```
            {
```

```
                printBoard();
```

```
                winningMessage(player);
```

```
                gameOver=true;
```

```
}
```

```
            if(checkDraw()) {
```

```
                printBoard();
```



```
        drawMessage();
        gameOver=true;
    }

    count++;
}

}

private static boolean checkDraw() {
    boolean res=false;

    if(      b[0]!='-' &&
            b[1]!='-' &&
            b[2]!='-' &&
            b[3]!='-' &&
            b[4]!='-' &&
            b[5]!='-' &&
            b[6]!='-' &&
            b[7]!='-' &&
            b[8]!='-'

        )
    res=true;

    return res;
}

private static void drawMessage() {
    System.out.println("Opps!! No more moves... ");
    System.out.println("-----");
    System.out.println(" It is a Draw ");
    System.out.println("-----");
    System.out.println(" Game Over ");
    System.out.println(" ----- ");

}
```



```
private static void winningMessage(int player) {  
    System.out.println("Congratulations... ");  
    System.out.println("-----");  
    System.out.println(" Player "+player+" Won!! ");  
    System.out.println("-----");  
    System.out.println(" Game Over ");  
    System.out.println(" ----- ");  
  
}  
  
private static int checkWinner(int player) {  
    int res=0;  
    switch(player) {  
        case 1:if(      b[0]=='X' && b[1]=='X' && b[2]=='X' ||  
                        b[3]=='X' && b[4]=='X' && b[5]=='X' ||  
                        b[6]=='X' && b[7]=='X' && b[8]=='X' ||  
  
                        b[0]=='X' && b[3]=='X' && b[6]=='X' ||  
                        b[1]=='X' && b[4]=='X' && b[7]=='X' ||  
                        b[2]=='X' && b[5]=='X' && b[8]=='X' ||  
  
                        b[0]=='X' && b[4]=='X' && b[8]=='X' ||  
                        b[2]=='X' && b[4]=='X' && b[6]=='X'  
  
                )  
        res=1;  
        break;  
        case 2:if(      b[0]=='O' && b[1]=='O' && b[2]=='O' ||  
                        b[3]=='O' && b[4]=='O' && b[5]=='O' ||  
                        b[6]=='O' && b[7]=='O' && b[8]=='O' ||  
  
                        b[0]=='O' && b[3]=='O' && b[6]=='O' ||  
                        b[1]=='O' && b[4]=='O' && b[7]=='O' ||  
                        b[2]=='O' && b[5]=='O' && b[8]=='O' ||  
  
                        b[0]=='O' && b[4]=='O' && b[8]=='O' ||  
                        b[2]=='O' && b[4]=='O' && b[6]=='O'  
  
                )  
    }  
}
```



```
res=2;
break;

}

return res;
}

private static boolean play(int player) {
    boolean res=true;

    symbol=player==1 ? 'X':'O';
    System.out.println("Player " +player+ " Turn ... ");
    int input=getInput();

    if(input==1 && b[0]=='-') {
        b[0]=symbol;
    }
    else if(input==2 && b[1]=='-') {
        b[1]=symbol;
    }
    else if(input==3 && b[2]=='-') {
        b[2]=symbol;
    }
    else if(input==4 && b[3]=='-') {
        b[3]=symbol;
    }
    else if(input==5 && b[4]=='-') {
        b[4]=symbol;
    }
    else if(input==6 && b[5]=='-' ) {
        b[5]=symbol;
    }
    else if(input==7 && b[6]=='-') {
        b[6]=symbol;
    }
    else if(input==8 && b[7]=='-') {
        b[7]=symbol;
    }
    else if(input==9 && b[8]=='-') {
        b[8]=symbol;
    }
}
```



```
    }
    else {
        res=false;
    }

    return res;
}

private static int getInput() {
    System.out.println("Enter Input[1-9]:");
    return sc.nextInt();
}

private static void printBoard() {
    // TODO Auto-generated method stub

    System.out.println(" Tic Tac Toe Game ");
    System.out.println("Player 1 = X, Player 2 = O ");
    System.out.println(" -----");
    System.out.println(" "+b[0]+" | "+b[1]+" | "+b[2]+" ");
    System.out.println(" -----");
    System.out.println(" "+b[3]+" | "+b[4]+" | "+b[5]+" ");
    System.out.println(" -----");
    System.out.println(" "+b[6]+" | "+b[7]+" | "+b[8]+" ");
    System.out.println(" -----");

}
}
```



Game Output:

Tic Tac Toe Game

Player 1 = X, Player 2 = O

- | - | -

- | - | -

- | - | -

Player 1 Turn ...

Enter Input[1-9]:

1

Tic Tac Toe Game

Player 1 = X, Player 2 = O

X | - | -

- | - | -

- | - | -

Player 2 Turn ...

Enter Input[1-9]:

2

Tic Tac Toe Game

Player 1 = X, Player 2 = O

X | O | -

- | - | -

- | - | -

Player 1 Turn ...

Enter Input[1-9]:

■ ප්‍රකාශනයේවාටිදී ගුණපුළුතා තරුණයෙන්



3

Tic Tac Toe Game

Player 1 = X, Player 2 = O

X | O | X-----
- | - | ------
- | - | -

Player 2 Turn ...

Enter Input[1-9]:



▣ Snake Game

Game Features:

1. ဂိမ်းဘုတ် (Board)
 - WIDTH = 20, HEIGHT = 16 ဖြင့် 20x16 ကရစ်ကွက်ကို ဖန်တီးထားပါတယ်။
 - Q = မြေခေါင်း, O = မြေကိုယ်, @ = အစာ။
2. မြေနဲ့ အစာ
 - မြေကို game board လယ်တည့်တည့် မှာ ချထားပါမယ်။
 - အစာကို ကျပန်းနေရာ မှာ ထူတ်ပေးပါမယ်။
3. ဂိမ်းလုပ်ဆောင်ချက်များ
 - clearScreen() : game board ကို refresh လုပ်ပါမယ်။
 - update() : လူပြေားမှုတွေကို အမြဲ update လုပ်ပါမယ်။
 - draw() : ဂိမ်းဘုတ်၊ အစာ နဲ့ မြေ တွေကို ဆွဲပါမယ်။
4. ထိန်းချုပ်မှု
 - W : အထက်, S : အောက်, A : ဘယ်, D : ညာ, Q : ထွက်။

Code:

```
package games;
```

```
import java.awt.event.KeyAdapter;  
import java.awt.event.KeyEvent;  
import java.io.IOException;
```

```
import javax.swing.JFrame;
```

■ မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



```
public class SnakeGame extends JFrame {  
  
    public static boolean gameOver=false;  
    public static int width=20;  
    public static int height=16;  
    public static int score=0;  
    public static char dir='0';  
  
    public SnakeGame(){  
  
        this.setTitle("Snake Game ");  
        this.setLayout(null);  
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);  
  
        this.setSize(400,100);  
        this.addKeyListener(new KeyAdapter) {  
  
            public void keyPressed(KeyEvent e) {  
                dir=e.getKeyChar();  
            }  
        });  
  
        this.setVisible(true);  
    }  
  
    public static void main(String[] args) throws InterruptedException, IOException {  
        // TODO Auto-generated method stub  
        new SnakeGame1();  
  
        Snake s=new Snake(width,height);  
        Board b=new Board(width,height);  
        Food f=new Food(width,height);  
  
        while(!gameOver) {  
  
            clearScreen();  
            update(s,b,f,dir,width,height);  
            draw(s,b,f,gameOver);  
        }  
    }  
}
```



```
    }

}

private static void update(Snake s, Board b, Food f, char dir, int width, int height) {
    // TODO Auto-generated method stub
    if(s.L>0) {

        //swapping new head position and tails position
        for(int i=s.L-1;i>=1;i--) {

            s.tailX[i]=s.tailX[i-1];
            s.tailY[i]=s.tailY[i-1];
        }

        s.tailX[0]=s.x;
        s.tailY[0]=s.y;
    }

    switch(dir) {

        case 'a':s.x--;break;
        case 'd':s.x++;break;
        case 'w':s.y--;break;
        case 's':s.y++;break;

    }

    if(s.x==f.x && s.y==f.y) {

        s.eatFood();
        f.reset(width, height, s);
        score=score+10;

    }

}
```

```
private static void draw(Snake s, Board b, Food f, boolean gameOver) {
```



```
// top bar
for(int i=0;i< width+2;i++) {
    System.out.printf("#");
    System.out.printf("\n");

//body drawing
for(int i=0;i< height;i++) {
    for(int j=0;j< width+2;j++) {

        if(j==0) {
            System.out.print("#");
        }
        else if(j==width+1) {
            System.out.print("#");
        }
        else if(s.x==j && s.y==i) {
            System.out.print("Q");
        }
        else if(f.x==j && f.y==i) {
            System.out.print("F");
        }
        else {

            boolean isTail=false;

            for(int k=0;k<s.L;k++) {
                if(s.tailX[k]==j && s.tailY[k]==i) {
                    System.out.print("O");
                    isTail=true;
                }
            }

            if(!isTail) {
                System.out.print(" ");
            }
        }
    }
}

System.out.println();
```



```
    }

    // bottom bar
    for(int i=0;i< width+2;i++)
        System.out.printf("#");
    System.out.printf("\n");

    System.out.println(" ===== ");
    System.out.println("Score :" + score);
    System.out.println(" ===== ");

}

private static void clearScreen() throws InterruptedException, IOException {
    // TODO Auto-generated method stub
    new ProcessBuilder("cmd","/c","cls").inheritIO().start().waitFor();
}

}

class Snake{

    int x;
    int y;
    int L;
    int[] tailX,tailY;

    public Snake(int width,int height) {
        this.x=width/2;
        this.y=height/2;
        this.L=0;
        this.tailX=new int[width*height];
        this.tailY=new int[width*height];
    }

    public void eatFood() {
        this.L++;
    }
}
```



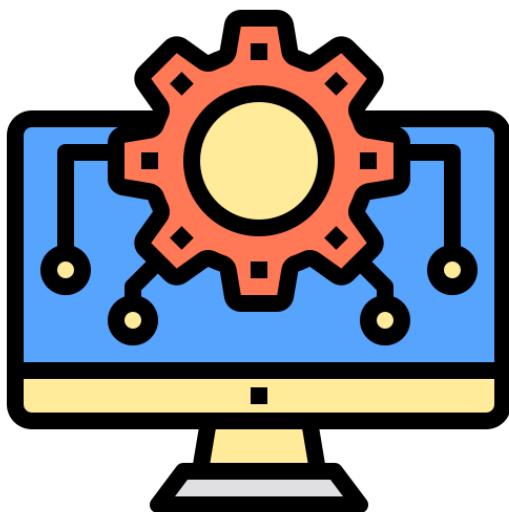
}

```
class Board{  
    int width;  
    int height;  
    public Board(int width,int height) {  
        this.width=width;  
        this.height=height;  
    }  
}
```

```
class Food{  
    int x;  
    int y;  
  
    public Food(int width,int height) {  
        this.x=(int) (Math.random() * width+1);  
        this.y=(int) (Math.random() * height);  
    }  
  
    public void reset(int width,int height,Snake s) {  
        this.x=(int) (Math.random()*width+1);  
        this.y=(int) (Math.random()*height);  
  
        if ((this.x == s.x) && (this.y == s.y)) {  
            s.eatFood();  
            reset(width, height, s);  
        }  
    }  
}
```



Algorithms



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



ALGORITHMS

Algorithm (အယ်လ်ဂိုရီသမ) ဆိုတာဘာလဲဆိုတော့ ပြဿနာတစ်ခုကိုဖြေရှင်းဖို့ အဆင့်ဆင့်ဆွဲနိုင် ကြားထားတဲ့ လုပ်ငန်းစဉ် ဖြစ်ပါတယ်။ ဂွန်ပျူးတာသိပို့နဲ့ သံချုပ်မှာ အခြေခံကျတဲ့ အရာဖြစ်ပြီး လုပ်ငန်းတိုင်းအတွက် Algorithm တစ်ခုရှိပါတယ်။

□ Algorithm ၏ အခြေခံလက္ခဏာများ

1. Clear and Unambiguous Steps - အဆင့်တိုင်းက ရှင်းလင်းရပါမယ်။ မရှင်းလင်းမှာ လုံးဝမရှိရပါ။
2. Well-defined Inputs/Outputs - ထည့်ပေးရမယ့်အချက်နဲ့ ရလာမယ့်အဖြေရှိရပါမယ်။
3. Finiteness - အဆုံးသတ်နိုင်တဲ့ လုပ်ငန်းစဉ်ဖြစ်ရပါမယ်။
4. Feasibility - လက်တွေ့လုပ်ဆောင်နိုင်တဲ့ အဆင့်များသာဖြစ်ရပါမယ်။
5. Language Independent - ဘယ် Programming Language နဲ့မဆို အကောင်အထည်ဖော်နိုင်ပါတယ်။

□ Algorithm ၏ ဥပမာများ

1. ရှိုးရှင်းသော Algorithm (ကိန်းဂဏန်းနှစ်ခုပေါင်းခြင်း)

1. Start
2. num1 နဲ့ num2 ကိုရယူပါ
3. sum = num1 + num2
4. sum ကိုပြုပါ
5. End

□ ဥပမာ

```
public class AddNumbers {  
    public static void main(String[] args) {  
        int num1 = 5;  
        int num2 = 7;  
        int sum = num1 + num2;  
    }  
}
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
        System.out.println("Sum is: " + sum);
    }
}
```

2. အဆင့်မြင့် Algorithm (Binary Search)

1. Start
2. Sorted array နဲ့ target value ကိုရယူပါ
3. low = 0, high = array.length - 1
4. While low <= high:
 - a. mid = (low + high) / 2
 - b. If array[mid] == target, return mid
 - c. If array[mid] < target, low = mid + 1
 - d. Else high = mid - 1
5. Return -1 (Not found)
6. End

▣ ဥပမာ

```
public class BinarySearch {
    public static int search(int[] arr, int target) {
        int left = 0;
        int right = arr.length - 1;

        while (left <= right) {
            int mid = left + (right - left) / 2;

            if (arr[mid] == target) {
                return mid;
            }

            if (arr[mid] < target) {
                left = mid + 1;
            } else {
                right = mid - 1;
            }
        }
    }
}
```



```
        return -1;
    }

public static void main(String[] args) {
    int[] data = {2, 5, 8, 12, 16, 23, 38, 56, 72, 91};
    int target = 23;
    int result = search(data, target);

    if (result == -1) {
        System.out.println("Element not found");
    } else {
        System.out.println("Element found at index: " + result);
    }
}
```

□ Algorithm အမျိုးအစားများ

1. Sorting Algorithms (စီခြင်း)

- Bubble Sort, Quick Sort, Merge Sort တိုပါဝင်ပါတယ်
- ဥပမာ: ကျောင်းသားအမှတ်များကို အမြင့်ဆုံးမှ အနိမ့်ဆုံးအလိုက်စီခြင်း

2. Searching Algorithms (ရှာဖွေခြင်း)

- Linear Search, Binary Search တိုပါဝင်ပါတယ်
- ဥပမာ: ဖုန်းစာရင်းထဲမှ အမည်တစ်ခုရှာခြင်း

3. Graph Algorithms (ဂရပ်စိသိမ္မာ)

- Dijkstra's Algorithm, BFS, DFS တိုပါဝင်ပါတယ်
- ဥပမာ: Google Maps မှ အမြန်ဆုံးလမ်းကြောင်းရှာခြင်း

4. Dynamic Programming (ဒိုင်းနမစ်ပရိုဂရမ်းမင်း)

- Fibonacci Sequence, Knapsack Problem တိုပါဝင်ပါတယ်
- ဥပမာ: ရင်းနှီးမြှုပ်နှံမှုမှ အမြတ်အများဆုံးရှာခြင်း



□ Algorithm ၏ အရေးပါမှု

1. အချိန်အမျှ - ကောင်းမွန်တဲ့ Algorithm ၏ processing time ကိုလျော့ချိန်ပါတယ်
2. Resource Optimization - Memory & computing power ကိုထိရောက်စွာအသုံးချိန်ပါတယ်
3. Scalability - Data ပမာဏများလာလည်း ကောင်းစွာအလုပ်လုပ်နိုင်ပါတယ်
4. Problem Solving - ရှုပ်ထွေးတဲ့ပြဿနာများကိုဖြေရှင်းနိုင်ပါတယ်

□ Algorithm Design Techniques

1. Divide and Conquer - ပြဿနာကိုအပိုင်းပိုင်းခွဲပြီး ဖြေရှင်းခြင်း (ဥပမာ - Merge Sort)
2. Greedy Method - လက်ရှိအခြေအနေမှာ အကောင်းဆုံးရွေးချယ်မှုကိုလုပ်ခြင်း
3. Dynamic Programming - Subproblems များကိုဖြေရှင်းပြီး ရလဒ်များကိုသိမ်းဆည်းခြင်း
4. Backtracking - ဖြစ်နိုင်ခြေအားလုံးကိုစမ်းကြည့်ပြီး မှားလျင်နောက်ပြန်ဆုတ်ခြင်း

□ Algorithm Analysis

Algorithm များကိုအဓိကအားဖြင့် မျိုးနဲ့ဆန်းစစ်ပါတယ်။

1. Time Complexity - အလုပ်လုပ်ရန် လိုအပ်သောအချိန်
 - $O(1)$ - Constant Time (အကောင်းဆုံး)
 - $O(n)$ - Linear Time
 - $O(n^2)$ - Quadratic Time (အဆုံးဆုံး)
2. Space Complexity - အလုပ်လုပ်ရန် လိုအပ်သော memory ပမာဏ



ဥပမာ: Linear Search vs Binary Search

Algorithm	Time Complexity	Space Complexity
Linear Search	$O(n)$	$O(1)$
Binary Search	$O(\log n)$	$O(1)$

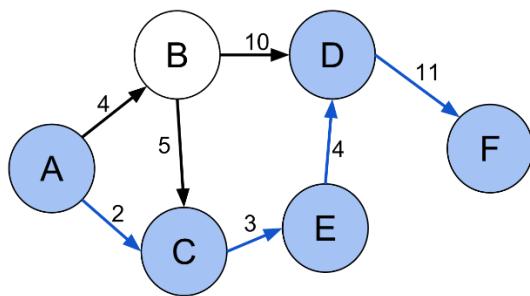
□ Real-world Applications

1. Social Media - သင့်အတွက်အဆင်ပြေမယ့် Post များစီစဉ်ခြင်း
2. E-commerce - သင့်စိတ်ဝင်စားနိုင်မယ့် ပစ္စည်းများအကြံပြေခြင်း
3. Healthcare - DNA Sequencing နဲ့ ရောဂါရာဖွေခြင်း
4. Finance - Stock Market ခန်းမှန်းခြင်းနဲ့ Fraud Detection

Algorithm များသည် ကျွန်ုပ်တို့နေစဉ်အသုံးပြုနေသော နည်းပညာများ၏ နောက်ကွယ်မှ အလုပ်လုပ်ပုံများဖြစ်ပါတယ်။ ကောင်းမွန်သော Algorithm တစ်ခုသည် ပြဿနာများကို ထိရောက်စွာနဲ့ မြန်ဆန်စွာ ဖြေရှင်းနိုင်ပါတယ်။ Algorithm များကိုနားလည်ခြင်းဖြင့် ပိုမိုကောင်းမွန်သော software များနဲ့ system များကိုတည်ဆောက်နိုင်ပါတယ်။

□ Dijkstra's Algorithm (Shortest Paths)

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



Dijkstra's Algorithm (ဒစ်ဂျက်စထရာ အယ်လ်ဂိုရီသမ်) ဆိုတာက Graph တစ်ခုတဲ့မှာ Node တစ်ခုကနေ အခြား Node တွေဆိုကို အချိန်အနည်းဆုံး/အကွာအဝေးအနည်းဆုံးနဲ့ ရောက်အောင် ရာဖွေပေးတဲ့ Algorithm ဖြစ်ပါတယ်။ 1956 ခုနှစ်မှာ Edsger W. Dijkstra ဆိုသူက တို့ထွင်ခဲ့တာဖြစ်ပါတယ်။

□ Dijkstra's Algorithm ၏ အခြေခံလုပ်ဆောင်ချက်

1. Shortest Path Problem ကိုဖြေရှင်းပေးပါတယ်
2. Weighted Graph တွေအတွက်အသုံးဝင်ပါတယ် (အနုတ်တန်ဖိုး negative weight မပါဝင်ရပါ)
3. Greedy Approach ကိုအသုံးပြုပါတယ် (လက်ရှိအခြေအနေမှာ အကောင်းဆုံးရွေးချယ်မှုကိုလုပ်ပါတယ်)

□ အလုပ်လုပ်ပုံ အဆင့်ဆင့်

1. Start
2. အားလုံး Node တွေကို အကန်အသတ်မရှိ (∞) distance သတ်မှတ်ပါ
3. စတင်မယ့် Node ကို distance = 0 သတ်မှတ်ပါ
4. မလည်ပတ်ရသေးတဲ့ Node တွေထဲက အနည်းဆုံး distance ရှိတဲ့ Node ကိုရွေးပါ (current node)
5. Current node နဲ့ ချိတ်ဆက်ထားတဲ့ အနီးချင်း Node တွေကို Update လုပ်ပါ:
 - a. New distance = current node distance + edge weight
 - b. New distance < existing distance ဖြစ်ရင် Update လုပ်ပါ
6. Current node ကို visited အဖြစ်မှတ်သားပါ
7. မလည်ပတ်ရသေးတဲ့ Node တွေရှိနေသရွှေ့ အဆင့် 4 ကိုပြန်လုပ်ပါ
8. End

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



ကျမ်း

```
import java.util.*;  
  
public class DijkstraAlgorithm {  
    static class Edge {  
        int target;  
        int weight;  
  
        Edge(int target, int weight) {  
            this.target = target;  
            this.weight = weight;  
        }  
    }  
  
    public static void dijkstra(List<List<Edge>> graph, int start) {  
        int n = graph.size();  
        int[] distances = new int[n];  
        boolean[] visited = new boolean[n];  
  
        Arrays.fill(distances, Integer.MAX_VALUE);  
        distances[start] = 0;  
  
        PriorityQueue<Edge> pq = new PriorityQueue<>(Comparator.comparingInt(e -> e.weight));  
        pq.add(new Edge(start, 0));  
  
        while (!pq.isEmpty()) {  
            Edge current = pq.poll();  
            int u = current.target;  
  
            if (visited[u]) continue;  
            visited[u] = true;  
  
            for (Edge edge : graph.get(u)) {  
                int v = edge.target;  
                int weight = edge.weight;  
  
                if (distances[u] + weight < distances[v]) {  
                    distances[v] = distances[u] + weight;  
                }  
            }  
        }  
    }  
}
```



```
        pq.add(new Edge(v, distances[v]));
    }
}

// Print shortest distances
System.out.println("Shortest distances from node " + start + ":");
for (int i = 0; i < n; i++) {
    System.out.println("To node " + i + ": " + distances[i]);
}
}

public static void main(String[] args) {
    // Example graph
    int n = 5; // Number of nodes
    List<List<Edge>> graph = new ArrayList<>();

    for (int i = 0; i < n; i++) {
        graph.add(new ArrayList<>());
    }

    // Add edges (node, weight)
    graph.get(0).add(new Edge(1, 4));
    graph.get(0).add(new Edge(2, 1));
    graph.get(1).add(new Edge(3, 1));
    graph.get(2).add(new Edge(1, 2));
    graph.get(2).add(new Edge(3, 5));
    graph.get(3).add(new Edge(4, 3));

    dijkstra(graph, 0); // Start from node 0
}
}
```

ရလဒ်:

Shortest distances from node 0:

To node 0: 0

To node 1: 3

To node 2: 1

- မြန်မာပြည်သားတိုင်း ကွန်ပျူဗာ တတ်ရမည်။



To node 3: 4

To node 4: 7

□ Dijkstra's Algorithm ၏ အသုံးဝင်မှုများ

1. မြေပုံအပလီကေးရှင်းများ - Google Maps, Waze တို့မှာ အတိုဆုံးလမ်းကြောင်းရှာဖွေခြင်း
2. Network Routing - ဒေတာပို့လွှတ်ရာမှာ အကောင်းဆုံးလမ်းကြောင်းရွေးချယ်ခြင်း
3. Traffic Management - ယဉ်ကြေပိတ်ဆိုမှုကိုရှောင်ရှားသည့်လမ်းကြောင်းများရှာဖွေခြင်း
4. Electricity Distribution - ဓာတ်အားဖြန့်ဖြူးရာမှာ အကောင်းဆုံးလိုင်းရွေးချယ်ခြင်း

□ Dijkstra's Algorithm ၏ ကန်သတ်ချက်များ

1. Negative Weights မလုပ်နိုင်ပါ - Graph ထဲမှာ negative weight တွေပါရင် မှားယွင်းတဲ့ရလဒ်ထွက်နိုင်ပါတယ်
2. Dense Graph တွေမှာ နှေးနိုင်ပါတယ် - Node အများကြီးပါတဲ့ Graph တွေအတွက် အခြား Algorithm တွေက ပုံမြန်နိုင်ပါတယ်
3. Path ကိုတွက်ရာမှာ Priority - အတိုဆုံး distance ကိုသာတွက်ပြီး အနည်းဆုံး Node အရေအတွက်နဲ့သွားတာမျိုးမဟုတ်ပါ

□ Time Complexity Analysis

- Binary Heap အသုံးပြုပါက: $O((V+E) \log V)$
 - V = Node အရေအတွက် (Vertices)
 - E = Edge အရေအတွက်
- Fibonacci Heap အသုံးပြုပါက: $O(E + V \log V)$

မှတ်ချက်: Java မှာ PriorityQueue က Binary Heap အသွင်နဲ့အလုပ်လုပ်ပါတယ်

□ Bellman-Ford vs Dijkstra

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။

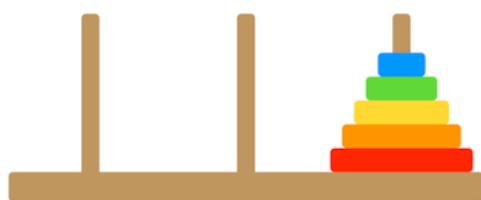


Feature	Dijkstra's Algorithm	Bellman-Ford Algorithm
Negative Weights	မလုပ်နိုင်ပါ	လုပ်နိုင်ပါတယ်
Time Complexity	$O((V+E) \log V)$	$O(VE)$
Type	Greedy	Dynamic Programming
Use Case	No negative weights	With negative weights

Dijkstra's Algorithm သည် လမ်းကြောင်းရှာဖွေရေးပြဿနာများအတွက် အလွန်ထိရောက်သော Algorithm တစ်ခုဖြစ်ပြီး Computer Science နယ်ပယ်တွင် အခြေခံကျသော Algorithm တစ်ခုအဖြစ် သင်ကြားလေ့ရှိပါတယ်။

ဖြစ်ပါတယ်။ Dijkstra's Algorithm ကို နားလည်ထားခြင်းဖြင့် Graph နှုန်းတဲ့ပြဿနာတွေကို ပိုမိုကောင်းမွန်စွာ ဖော်ရှင်းနိုင်မည်ဖြစ်ပါတယ်။

▣ Tower of Hanoi Algorithm



- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



Tower of Hanoi (ဟာနိုင်းရွှေကျောက်တိုင်) Algorithm ဆိုတာက ပဟောဌာနပြဿနာတစ်ခုကို အဆင့်ဆင့်ဖြေရှင်းနည်းဖြစ်ပါတယ်။ ပြင်သစ်သချုပ်ပညာရှင် Édouard Lucas ၂ ၁၈၈၃ ခုနှစ်မှာ စတင်မိတ်ဆက်ခဲ့တာဖြစ်ပါတယ်။

□ Tower of Hanoi ပဟောဌာန အခြေခံအစိတ်အပိုင်းများ

1. တိုင်အရှည် ၃ခု (အများအားဖြင့် A, B, C လို့ခေါ်ပါတယ်)
2. အရွယ်အစားအမျိုးမျိုးရှိ ပိုင်အခွေများ (အရွယ်အစားငယ်တဲ့အခွေက အပေါ်ဆုံးမှာရှိပါတယ်)
3. စည်းမျဉ်း ၃ချက်:
 - o တစ်ကြိမ်တွင် အခွေတစ်ခုသာရွှေ့ရပါမယ်
 - o အပေါ်ဆုံးအခွေကိုသာ ရွှေ့နိုင်ပါတယ်
 - o အကြီးအခွေကို အသေးအခွေပေါ်မတင်ရပါ

□ Algorithm ၏ အဆင့်ဆင့်လုပ်ဆောင်ချက် (Recursive Approach)

1. Start
2. n = အခွေအရေအတွက်
3. source = မူလတိုင် (A)
4. target = ပစ်မှတ်တိုင် (C)
5. auxiliary = အကူလိုင် (B)
6. Hanoi(n, source, target, auxiliary)
7. End

Hanoi(n, source, target, auxiliary):

```
IF n == 1 THEN
    "Move disk 1 from source to target"
ELSE
    Hanoi(n-1, source, auxiliary, target)
    "Move disk n from source to target"
    Hanoi(n-1, auxiliary, target, source)
```

□ ဥပမာ

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
public class TowerOfHanoi {  
    public static void solveHanoi(int n, char source, char target, char auxiliary) {  
        if (n == 1) {  
            System.out.println("Move disk 1 from " + source + " to " + target);  
        } else {  
            solveHanoi(n - 1, source, auxiliary, target);  
            System.out.println("Move disk " + n + " from " + source + " to " + target);  
            solveHanoi(n - 1, auxiliary, target, source);  
        }  
    }  
  
    public static void main(String[] args) {  
        int numberOfDisks = 3; // အစွဲ ရှုနဲ့မြတ်ပါမယ်  
        solveHanoi(numberOfDisks, 'A', 'C', 'B'); // A ဘိုင်ကနေ C ဘိုင်ကိုရွှေ့မယ်၊ B ဘိုင်ကအကူ  
    }  
}
```

ရလဒ် (အခွဲ ရွှေ့အတွက်):

Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C

- Algorithm ၏ အရေးကြီးသောအချက်များ

1. Recursive Nature - ပြဿနာကိုယ်တိုင်ကိုယ်ကျ ထပ်ခေါ်သောနည်းလမ်း (Divide and Conquer)

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



2. Minimum Moves - အခွဲ n ခုအတွက် အနည်းဆုံး $2^n - 1$ ကြိမ်ချေရပါမယ်

- အခွဲ 3 ခုဆို 7 ကြိမ်
- အခွဲ 4 ခုဆို 15 ကြိမ်

3. Time Complexity - $O(2^n)$ (Exponential time)

□ လက်တွေ့အသုံးချမှုများ

1. Recursion ကိုသင်ကြားရာတွင် - Algorithm design အတွက် အကောင်းဆုံးဥပမာ
2. Memory Management - Stack လုပ်ဆောင်ပုံကိုနားလည်ရန်
3. Puzzle Solving - ဆင်ခြင်တုတရားဖွံ့ဖြိုးရေး
4. Backup Systems - ဒေတာသိမ်းဆည်းမှုစနစ်များတွင် အသုံးချ

□ ဥပမာအဆင့်ဆင့်ရှင်းလင်းချက် ($n=3$)

1. အကြိုးဆုံးအခွဲ (3) ကို C ကိုရွှေ့ဖို့
 - အပေါ်က အခွဲ J၉ (1,2) ကို B ကိုအရင်ရွှေ့ရမယ်
 - အခွဲ 1 ကို A → C
 - အခွဲ 2 ကို A → B
 - အခွဲ 1 ကို C → B
2. အခွဲ 3 ကို A → C
3. B ပေါ်က အခွဲ J၉ကို C ကိုရွှေ့မယ်
 - အခွဲ 1 ကို B → A
 - အခွဲ 2 ကို B → C
 - အခွဲ 1 ကို A → C

□ Iterative Solution (Loop အသုံးပြုနည်း)

```
import java.util.Stack;  
  
public class TowerOfHanoiIterative {
```

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
class Move {  
    int n;  
    char source, target, auxiliary;  
    boolean isProcessed;  
  
    Move(int n, char s, char t, char a) {  
        this.n = n;  
        this.source = s;  
        this.target = t;  
        this.auxiliary = a;  
    }  
}  
  
public static void solveHanoi(int n, char source, char target, char auxiliary) {  
    Stack<Move> stack = new Stack<>();  
    stack.push(new Move(n, source, target, auxiliary));  
  
    while (!stack.isEmpty()) {  
        Move current = stack.pop();  
  
        if (current.n == 1) {  
            System.out.println("Move disk 1 from " + current.source + " to " + current.target);  
        } else {  
            if (!current.isProcessed) {  
                current.isProcessed = true;  
                stack.push(current);  
                stack.push(new Move(current.n-1, current.source, current.auxiliary, current.target));  
            } else {  
                System.out.println("Move disk " + current.n + " from " + current.source + " to " + current.target);  
                stack.push(new Move(current.n-1, current.auxiliary, current.target, current.source));  
            }  
        }  
    }  
}  
  
public static void main(String[] args) {  
    solveHanoi(3, 'A', 'C', 'B');  
}
```

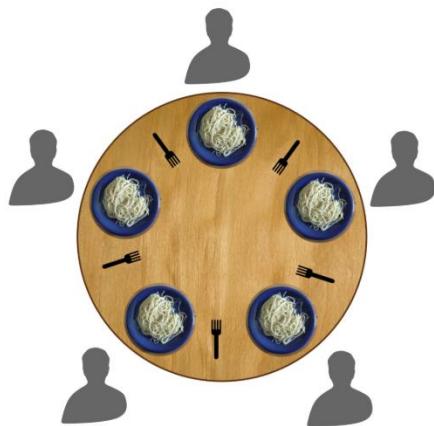


□ သိမှတ်ဖွယ်ရာများ

1. သမိုင်းကြောင်း - စီယက်နစ်နိုင်ငံရှိ ဟန္တိုင်းမြို့မှ ဘန်းကြီးများ ရွှေ့ပြောင်းခြင်းအကြောင်း ဒဏ္ဍာရီမှဆင်းသက်လာပါတယ်
2. အခွဲ 64ခုအတွက် - ရွှေ့ရမယ့်အကြိမ်ရေ 18,446,744,073,709,551,615 (ကမ္ဘာပျက်သည်အထိကြာမယ်လိုအပါတယ်!)
3. သင်ကြားရေးဆိုင်ရာ - Recursion ကိုနားလည်ရန် အကောင်းဆုံးသုတေသန

Tower of Hanoi Algorithm သည် recursive thinking နှင့် problem decomposition ကိုလေ့လာရန် အကောင်းဆုံးနမူနာတစ်ခုဖြစ်ပါတယ်။ ဒါ Algorithm ကို နားလည်ထားခြင်းဖြင့် ရှုပ်ထွေးသော recursive problem များကို ကောင်းစွာဖြေရှင်းနိုင်မည်ဖြစ်ပါတယ်။

□ Five Dining Philosophers Problem



- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



Five Dining Philosophers Problem (ဒသနပညာရှင် ၅ဦး ညစာစားခြင်းပြဿနာ) ဆိတာက ကြားခံ အရင်းအမြစ်များ (shared resources) ကိုအသုံးပြုရေတွင် ဖြစ်ပေါ်လာနိုင်သော deadlock နှင့် starvation ပြဿနာများကိုလေ့လာရန် puzzle ဟနေ့တို့ တစ်ခု ဖြစ်ပါတယ်။ Edsger Dijkstra က 1965 ခုနှစ်မှာ စတင်မိတ်ဆက်ခဲ့တာဖြစ်ပါတယ်။

□ ပြဿနာ၏ အခြေခံအချက်များ

1. ဒသနပညာရှင် ၅ဦး - စားပွဲစိုင်းပေါ်တွင်ထိုင်နေကြပါတယ်
2. ခရင်း (fork) ငါးခု - စားပွဲနဲ့ကပ်လျက်တည်ရှိပါတယ် (ဒသနပညာရှင်တစ်ဦးနှင့်တစ်ဦးကြား ဘုံ)
3. ခေါက်ဆွဲပန်းကန် ၅ခု - တစ်ဦးတစ်ပန်းကန်

□ လုပ်ဆောင်ချက်များ

1. စဉ်းစားခြင်း - ဒသနပညာရှင်များသည် အချိန်အများစုကို စဉ်းစားနေကြပါတယ်
2. ထမင်းစားခြင်း - ဆာလောင်လာပါက အနီးနားရှိ ခရင်း နှစ်လက်ကိုကောက်ယူပြီး စားသုံးပါတယ်

□ ပြဿနာ၏ အခိုကအချက်

- Deadlock ဖြစ်နိုင်ခြေး: ဒသနပညာရှင်အားလုံးက ဘယ်ဘက်ခရင်း ကိုတစ်ပြိုင်နက်ကောက်ယူလိုက်ပါက ညာဘက်ခရင်း မရနိုင်တော့ဘဲ အားလုံးစားလို့မရတော့ပါ
- Starvation ဖြစ်နိုင်ခြေး: အချို့ဒသနပညာရှင်များ အစာမစားရဘဲ ကြာရှည်စွာစောင့်ဆိုင်းနေရနိုင်ပါတယ်

□ ဥပမာ

```
import java.util.concurrent.Semaphore;

class Philosopher extends Thread {
    private int id;
    private Semaphore leftFork;
    private Semaphore rightFork;
    private Semaphore table;

    public Philosopher(int id, Semaphore leftFork, Semaphore rightFork, Semaphore table) {
        this.id = id;
        this.leftFork = leftFork;
        this.rightFork = rightFork;
    }
}
```



```
this.table = table;  
}  
  
private void think() throws InterruptedException {  
    System.out.println("Philosopher " + id + " is thinking");  
    Thread.sleep((long)(Math.random() * 1000));  
}  
  
private void eat() throws InterruptedException {  
    System.out.println("Philosopher " + id + " is eating");  
    Thread.sleep((long)(Math.random() * 1000));  
}  
  
public void run() {  
    try {  
        while (true) {  
            think();  
  
            table.acquire(); // စားပွဲစဉ်ချင်ရှုခြင်း  
  
            leftFork.acquire(); // ဘယ်ဘက်ခရင်း ကောက်ယူခြင်း  
            System.out.println("Philosopher " + id + " picked up left fork");  
  
            rightFork.acquire(); // ညာဘက်ခရင်းကောက်ယူခြင်း  
            System.out.println("Philosopher " + id + " picked up right fork");  
  
            eat();  
  
            rightFork.release(); // ညာဘက်ခရင်းပြန်ချုခြင်း  
            System.out.println("Philosopher " + id + " put down right fork");  
  
            leftFork.release(); // ဘယ်ဘက်ခရင်းပြန်ချုခြင်း  
            System.out.println("Philosopher " + id + " put down left fork");  
  
            table.release(); // စားပွဲrelease ခွင့်ပြုခြင်း  
        }  
    } catch (InterruptedException e) {  
        System.out.println("Philosopher " + id + " was interrupted");  
    }  
}
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
}

}

public class DiningPhilosophers {
    static final int NUM_PHILOSOPHERS = 5;

    public static void main(String[] args) {
        Semaphore[] forks = new Semaphore[NUM_PHILOSOPHERS];
        for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
            forks[i] = new Semaphore(1); // ခရင်း တစ်လက်စီ
        }

        Semaphore table = new Semaphore(NUM_PHILOSOPHERS - 1); // အသနပညာရှင် ရှိုးသာ တစ်ပြို
        ငါက်စားနိုင်

        Philosopher[] philosophers = new Philosopher[NUM_PHILOSOPHERS];
        for (int i = 0; i < NUM_PHILOSOPHERS; i++) {
            Semaphore leftFork = forks[i];
            Semaphore rightFork = forks[(i + 1) % NUM_PHILOSOPHERS];

            philosophers[i] = new Philosopher(i, leftFork, rightFork, table);
            philosophers[i].start();
        }
    }
}
```

□ အဖြောနည်းများ

1. Resource Hierarchy Solution:
 - ခရင်း(fork) များကိုနံပါတ်တပ်ပြီး အမြတမ်းအနိမ့်ဆုံးနံပါတ်ရှိခြင်ကိုအရင်ကောက်ယူပါ
 - Deadlock ကိုရှောင်ရှားနိုင်ပါတယ်
2. Arbitrator Solution:
 - Waiter (သို့) Table Semaphore တစ်ခုထားပြီး အသနပညာရှင် ရှိုးသာ တစ်ပြိုင်နှင်းနှင့်ပါ
 - ကျွန်ုပ်တို့၏ Java code တွင် ဤနည်းလမ်းကိုအသုံးပြုထားပါတယ်
3. Timeout Solution:

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



- ခရင်း ကောက်ယူရန် အခိုင်သတ်မှတ်ထားပါက မရပါက လက်လွှတ်ပြီး
နောက်တစ်ကြိမ်ပြန်ကြိုးစားပါ

□ အရေးကြီးသော Concepts များ

1. Deadlock Prevention:

- Mutual Exclusion: ခရင်း များကို တစ်ချိန်တွင် အသုနပညာရှင်တစ်ဦးသာ အသုံးပြနိုင်ပါတယ်
- Hold and Wait: ခရင်း နှစ်လက်လုံးမရမချင်း စောင့်ဆိုင်းနေပါတယ်
- No Preemption: ခရင်း များကို အတင်းအကြပ်မယူနိုင်ပါ
- Circular Wait: အသုနပညာရှင်အားလုံးက ညာဘက်ခရင်း ကိုစောင့်နေကြပါတယ်

2. Starvation Avoidance:

- Fair scheduling algorithms များကိုအသုံးပြုပါ
- Random wait times များထည့်သွင်းပါ

□ Real-world Applications

1. Operating Systems - Process scheduling နှင့် resource allocation
2. Distributed Systems - Database locking mechanisms
3. Network Protocols - Channel allocation in communication networks
4. Multithreaded Applications - Thread synchronization

□ သိမှတ်ဖွယ်ရာများ

1. ပထမဆုံးဥပမာ - Concurrent programming ကိုလေ့လာရာတွင် အခြေခံကျသောဥပမာတစ်ခု
2. ပညာရေးဆိုင်ရာ - Deadlock နှင့် synchronization concepts များကိုသင်ကြားရာတွင် အသုံးဝင်ပါတယ်
3. Variations - အသုနပညာရှင်အရေအတွက်ကို ပြောင်းလဲချုပ်လည်း လေ့လာနိုင်ပါတယ်

ဖြစ်ပါတယ်။ Five Dining Philosophers Problem သည် concurrent systems တွင် resource sharing နှင့် synchronization ပြဿနာများကို နားလည်ရန် အကောင်းဆုံးနူးနူးနားတစ်ခုဖြစ်ပါတယ်။



Software Development Architectures



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



Java Software Development Architecture ဆိုတာ Java ပရိုဂရမ်းမင်းဘာသာစကားနဲ့ software application များ
တည်ဆောက်ရာမှာ အသုံးပြုတဲ့ ဒီဇိုင်းပုံစံများ ဖြစ်ပါတယ်။ ဒီ architecture တွေက ကုဒ်တွေကို စနစ်တကျ
စီမံခန့်ခွဲနိုင်ဖို့ ရေရှည်တည်တံ့တဲ့ ဆော်ဖဲတွေ ဖန်တီးနိုင်ဖို့ ကူညီပေးပါတယ်။

Layered Architecture က Java application တွေမှာ အသုံးအများဆုံး architecture တစ်ခု ဖြစ်ပါတယ်။ ဒီ architecture မှာ Presentation Layer (UI), Business Layer (Logic), Persistence Layer (Database) ဆိုပြီး
အလွှာသုံးလွှာ ခွဲထားပါတယ်။ Spring Framework နဲ့ ရေးသားတဲ့ application အများစုံမှာ ဒီ architecture ကို
အသုံးပြုကြပါတယ်။

Microservices Architecture က modern Java application တွေအတွက် အကောင်းဆုံး ရွှေးချယ်မှုတစ်ခု
ဖြစ်ပါတယ်။ Application ကို သီးခြား service ငယ်လေးတွေအဖြစ် ခွဲထုတ်ပြီး တစ်ခုချင်းစီကို သီးသန် develop နဲ့
deploy လုပ်နိုင်ပါတယ်။ Spring Boot နဲ့ Spring Cloud တို့က ဒီ architecture ကို အကောင်အထည်ဖော်ဖို့
အသောက်အကူပြုပါတယ်။

Event-Driven Architecture ကတော့ asynchronous processing နဲ့ real-time application တွေအတွက်
သင့်တော်ပါတယ်။ Components တွေအကြား event တွေပိုပြီး communicate လုပ်တဲ့အတွက် loosely coupled
system တွေ တည်ဆောက်နိုင်ပါတယ်။ Java မှာ Spring Framework ရဲ့ event handling mechanism နဲ့ Apache
Kafka တို့က ဒီ architecture ကို implement လုပ်ဖို့ အသုံးဝင်ပါတယ်။

Clean Architecture က long-term maintainability ကို အဓိကထားတဲ့ architecture ဖြစ်ပါတယ်။ Business logic
ကို core အဖြစ်ထားရှုပြီး framework နဲ့ external dependencies တွေကနေ လွတ်ကင်းအောင်
ဒီဇိုင်းခွဲထားပါတယ်။ Java မှာ dependency injection နဲ့ interface segregation principles တွေကို အသုံးပြုပြီး
implement လုပ်ကြပါတယ်။

Serverless Architecture က cloud computing နဲ့အတူ ခေတ်စားလာတဲ့ architecture တစ်ခု ဖြစ်ပါတယ်။ Java
functions တွေကို cloud platforms (AWS Lambda, Azure Functions) တွေမှာ deploy လုပ်ပြီး run နိုင်ပါတယ်။
Pay-as-you-go model ဖြစ်တဲ့အတွက် cost-effective ဖြစ်ပြီး auto-scaling feature တွေလည်း ပါဝင်ပါတယ်။

Java software architecture တွေကို ရွှေးချယ်ရာမှာ project requirements, team size, scalability needs နဲ့
maintenance considerations တွေကို အခြေခံသင့်ပါတယ်။ Layered Architecture က ရှိုးရှင်းတဲ့ application



တွေအတွက် ကောင်းမွန်ပြီး Microservices Architecture က complex, large-scale system တွေအတွက် ပိုသင့်တော်ပါတယ်။

Architecture ရွေးချယ်မှုက project ရဲ့ အောင်မြင်မှုကို ကြီးမားစွာ အထောက်အကူ ပြနိုင်တဲ့အတွက် ဂရတစိုက် စဉ်းစားသင့်ပါတယ်။ Modern Java applications အများစုတွင် Layered Architecture နှင့် Microservices Architecture တို့ကို အများဆုံးတွေ့ရပါတယ်။ ဒီစာအပ်မှာ လူသုံးအများဆုံး Layered Architecture design တွေဖြစ်တဲ့ Model-Repository-Services Architecture အကြောင်းနဲ့ Model-View-Controller Architecture အကြောင်းပြောပြုသွားပါမယ်။

□ Model-Repository-Services Architecture

Model-Repository-Service Architecture ဆိတ် modern Java application တွေမှာအသုံးများတဲ့ layered architecture တစ်ခုဖြစ်ပါတယ်။ ဒီ architecture မှာ application ကို Model, Repository, Service ဆိုပြီး အလွှာသုံးလွှာခွဲထားပါတယ်။

Model ကတော့ application ရဲ့ data structure နဲ့ business entities တွေကိုကိုယ်စားပြုပါတယ်။ Database table တစ်ခုစိတ်ကို Java class တစ်ခုအနေနဲ့ဖော်ပြပါတယ်။ JPA annotations တွေသုံးပြီး database schema နဲ့ mapping လုပ်လေ့ရှုပါတယ်။ ဥပမာ @Entity, @Table, @Column အစရှိတဲ့ annotations တွေကိုသုံးပါတယ်။

Repository layer က data access logic တွေအတွက်တာဝန်ယူပါတယ်။ Database နဲ့တိုက်ရှိက်အလုပ်လုပ်ပြီး CRUD (Create, Read, Update, Delete) operations တွေကိုလုပ်ဆောင်ပါတယ်။ Spring Data JPA သုံးရင် interface အနေနဲ့သတ်မှတ်ပြီး implementation ကို framework ကအလိုအလျောက်ဖန်တီးပေးပါတယ်။

Service layer မှာတော့ business logic တွေအားလုံးပါဝင်ပါတယ်။ Repository layer ကိုသုံးပြီး data တွေရယူကာ complex business rules တွေကိုအကောင်အထည်ဖော်ပါတယ်။ Transaction management တွေကိုလည်း service layer မှာလုပ်လေ့ရှုပါတယ်။ @Service annotation နဲ့သတ်မှတ်ပြီး controller ကနေခေါ်သုံးပါတယ်။

Controller layer ကိုတော့ ဒီ architecture မှာထည့်သွင်းဖော်ပြလေ့မရှုပေမယ့် အမှန်တကယ်တော့ရှုပါတယ်။ Controller က user requests တွေကိုလက်ခံပြီး service layer ကိုခေါ်သုံးကာ response ပြန်ပေးပါတယ်။ @RestController annotation သုံးပြီး REST API endpoints တွေကိုသတ်မှတ်လေ့ရှုပါတယ်။

ဒီ architecture ရဲ့အားသာချက်က separation of concerns ကောင်းမွန်တဲ့ဖြစ်ပါတယ်။ Data access logic နဲ့ business logic ကိုလုံးဝခွဲထားတဲ့အတွက် code တွေကိုပိုပြီးထိန်းသိမ်းရလွှာယ်ကူပါတယ်။ Unit testing လုပ်ရတာလည်းပိုလွှာယ်ကူပါတယ်။

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။



Spring Boot application တွေမှာ အခြား architecture ကိုအများဆုံးတွေ့ရပါတယ်။ Entity class တွေက model package အောက်မှာ၊ repository interfaces တွေက repository package အောက်မှာ၊ service classes တွေက service package အောက်မှာထားလေ့ရှိပါတယ်။

ဖြစ်ပါတယ်။ Model-Repository-Service Architecture က medium to large scale Java applications တွေအတွက်အထူးသင့်တော်ပါတယ်။ Layered approach ဖြစ်တဲ့အတွက် team members တွေအချင်းချင်းပူးပေါင်းအလုပ်လုပ်ရတာလွယ်ကူပြီး project တွေကိုလွယ်လွယ်ကူကူ scale လုပ်နိုင်ပါတယ်။

□ Model-View-Controller

MVC (Model-View-Controller) ဆိုတာ software development မှာအသုံးများတဲ့ design pattern တစ်ခုဖြစ်ပါတယ်။ ဒီ pattern မှာ application တစ်ခုကို Model, View, Controller ဆိုပြီး အပိုင်းသုံးစိုင်းခွဲထားပါတယ်။

Model က application ရဲ့ data နဲ့ business logic တွေကို ကိုယ်စားပြုပါတယ်။ Database နဲ့တိုက်ရှိက်အလုပ်လုပ်ပြီး data validation rules တွေ၊ business rules တွေပါဝင်ပါတယ်။ Java မှာ Model class တွေကို POJO (Plain Old Java Object) အနေနဲ့ ရေးလေ့ရှိပါတယ်။

View က user interface (UI) ကိုတာဝန်ယူပါတယ်။ Model ထဲက data တွေကို user မြင်ရအောင် ပြသပေးပါတယ်။ Web application တွေမှ HTML, CSS နဲ့ JavaScript တို့ကိုသုံးပြီး View တည်ဆောက်လေ့ရှိပါတယ်။ Java web framework တွေဖြစ်တဲ့ Spring MVC မှာ JSP, Thymeleaf နဲ့ FreeMarker တို့ကို View technology အဖြစ်သုံးကြပါတယ်။

Controller က user request တွေကိုလက်ခံပြီး Model နဲ့ View အကြား ချိတ်ဆက်ပေးပါတယ်။ User ကနေလာတဲ့ input တွေကိုလက်ခံပြီး လိုအပ်တဲ့ business logic တွေကို Model မှာခေါ်သုံးကာ သင့်တော်တဲ့ View ကိုရွေးချယ်ပြီး response ပြန်ပေးပါတယ်။ Java မှာ Controller တွေကို @Controller annotation နဲ့သတ်မှတ်လေ့ရှိပါတယ်။

MVC pattern ရဲ့ အဓိကအားသာချက်က separation of concerns ဖြစ်ပါတယ်။ အလွှာတစ်ခုစိုက သူ့တာဝန်နဲ့သူ အာရုံစိုကတဲ့အတွက် code တွေကို ပိုမိုထိန်းသိမ်းရလွယ်ကူဖော်ပါတယ်။ ဥပမာ UI ပြောင်းရင် View ကိုပြင်ရမှာဖြစ်ပြီး business logic ကိုမထိခိုက်ဖော်ပါဘူး။

Java web development မှာ Spring MVC framework က MVC pattern ကို အပြည့်အဝအားကိုးထားပါတယ်။ Spring Boot နဲ့တဲ့သုံးရင် MVC application တွေကို အလွယ်တကူတည်ဆောက်နိုင်ပါတယ်။ DispatcherServlet က



front controller အဖြစ်အလုပ်လုပ်ပြီး request တွေကို သက်ဆိုင်ရာ Controller method တွေဆီ ဦးတည်ပေးပါတယ်။

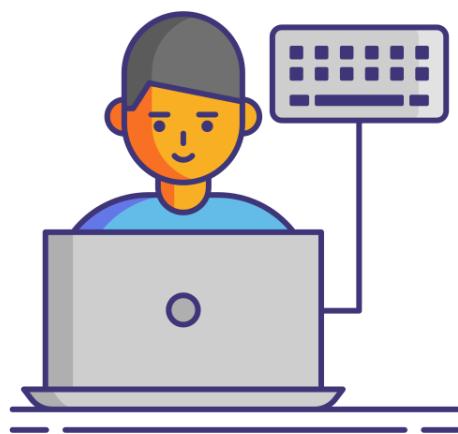
MVC pattern ကို web application တွေအပြင် desktop application တွေ၊ mobile app တွေမှာလည်းအသုံးပြနိုင်ပါတယ်။ JavaFX လို့ framework တွေမှာလည်း MVC pattern ကိုတွေ့ရတတ်ပါတယ်။ ဒါပေမယ့် web application တွေမှာတော့ အထူးသဖြင့် အသုံးများပါတယ်။

MVC မှာသတိထားရမယ့်အချက်က Controller မှာ business logic အများကြီးမရောက်သွားအောင် ဂရိစိုက်ရမှာဖြစ်ပါတယ်။ Business logic တွေကို Model ထဲမှာပတ္တားသင့်ပြီး Controller က request/response handling အတွက်သာ အာရုံစိုက်သင့်ပါတယ်။

ဖြစ်ပါတယ်။ MVC pattern က beginner developer တွေအတွက်နားလည်ရလွယ်ကူပြီး လက်တွေ့အသုံးဝင်တဲ့ architecture တစ်ခုဖြစ်ပါတယ်။ Java web development လေ့လာနေသူတိုင်း ဒီ design pattern ကိုနားလည်ထားသင့်ပါတယ်။



BEST PRACTICES



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



BEST PRACTICES

Java Developer တစ်ယောက်အနေနဲ့ Code Quality, Performance, Security နဲ့ Maintainability တို့ကို
အချင့်ကြပိုလိုပါတယ်။ ဒီမှာ အရေးကြီးတဲ့ Best Practices တွေကို ဖော်ပြပေးမယ်။

1. Clean Code ရေးပါ

- ✓ Meaningful Variable/Method Names သုံးပါ
- ✗ int a; String b; (မသုံးရ)
- ✓ int studentAge; String studentName; (သုံးရ)
- ✓ Single Responsibility Principle (SOLID) ကိုလိုက်နာပါ

```
// ✗ Bad - တစ်ခုထဲမှ အရာအားလုံးလုပ်နေတာ
public void processStudentData() {
    // Database Connection
    // Data Validation
    // Business Logic
    // File Saving
}
```

```
// ✓ Good - တစ်ခုချင်းစိတ် Method အသီးသီးခွဲထားတာ
public void connectToDatabase() {...}
public void validateData() {...}
public void applyBusinessLogic() {...}
public void saveToFile() {...}
```

2. Java 8+ Features တွေကိုသုံးပါ

- ✓ Lambda Expressions & Stream API

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5);
```

```
// ✗ Old Way (Loop)
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
for (int num : numbers) {  
    if (num % 2 == 0) {  
        System.out.println(num);  
    }  
}  
  
// ✅ Modern Way (Stream + Lambda)  
numbers.stream()  
.filter(n -> n % 2 == 0)  
.forEach(System.out::println);
```

✓ Optional Class (Null Safety)

```
// ✗ Bad (Null Check မလုပ်ရင် NPE ဖြစ်မယ်)  
String name = student.getName();  
if (name != null) {  
    System.out.println(name.toUpperCase());  
}  
  
// ✅ Good (Optional သုံးပြီး Null Safe ဖြစ်အောင်လုပ်)  
Optional<String> nameOpt = Optional.ofNullable(student.getName());  
nameOpt.ifPresent(n -> System.out.println(n.toUpperCase()));
```

3. Exception Handling ကိုမှန်မှန်ကန်ကန်လုပ်ပါ

- ✓ Specific Exceptions ကိုဖမ်းပါ
- ✗ catch (Exception e) (အလုယ်မဖမ်းရ)
- ✓ catch (IOException | SQLException e) (သက်ဆိုင်ရာ Exception တွေကိုပဲဖမ်းရ)

✓ Custom Exceptions သုံးပါ

```
public class InvalidStudentException extends RuntimeException {  
    public InvalidStudentException(String message) {  
        super(message);  
    }
```

- မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



```
    }  
}  
  
// သုတေသန  
if (student.getAge() < 0) {  
    throw new InvalidStudentException("Age cannot be negative!");  
}
```

4. Performance Optimization

- ✓ String Concatenation မှာ StringBuilder သုတေသန
- ✗ String result = str1 + str2 + str3; (Memory Inefficient)
- ☑ StringBuilder sb = new StringBuilder(); sb.append(str1).append(str2).append(str3);

✓ Collections ရွေးချယ်မှု

- ArrayList → Random Access များရင်
- LinkedList → Frequent Insertions/Deletions ရင်
- HashSet → Unique Values ငိုရင်

5. Testing & Debugging

✓ Unit Testing (JUnit 5)

```
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.*;  
  
class CalculatorTest {  
    @Test  
    void testAddition() {  
        Calculator calc = new Calculator();  
        assertEquals(5, calc.add(2, 3));  
    }  
}
```



{}

✓ Logging (SLF4J + Logback)

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class MyClass {
    private static final Logger logger = LoggerFactory.getLogger(MyClass.class);

    public void process() {
        logger.info("Processing started...");
        try {
            // Business Logic
        } catch (Exception e) {
            logger.error("Error occurred: ", e);
        }
    }
}
```

6. Security Practices

✓ SQL Injection ကာကွယ်ရှိ PreparedStatement သုံးပါ

✗ String query = "SELECT * FROM users WHERE username = '" + input + "'"; (Risk!)

☑ PreparedStatement ps = connection.prepareStatement("SELECT * FROM users WHERE username = ?");

✓ Sensitive Data စောက်လုပ်ရန် transient လုပ်ပါ

```
public class User implements Serializable {
    private String username;
    private transient String password; // Serialize မလုပ်ဘူး
}
```

7. Version Control & CI/CD

■ မြန်မာပြည်သားတိုင်း ဂွန်ပျူးတာ တတ်ရမည်။



- ✓ Git ကိုအခြေခံပြီး ကောင်းကောင်းသံဃားပါ

git commit -m "Meaningful Message"

git rebase vs git merge ကိုသိတဲ့ပါ

- ✓ Jenkins/GitHub Actions နဲ့CI/CD Pipeline တည်ဆောက်ပါ

8. ဆက်လက်လေ့လာရန်

- ✓ Design Patterns (Singleton, Factory, Observer)
- ✓ Spring Framework (Spring Boot, Spring Security)
- ✓ Microservices & Cloud (Docker, Kubernetes, AWS)

Short Notes:

- Clean Code ရေးပါ
- Java 8+ Features (Streams, Optional, Lambda) သံဃားပါ
- Testing & Logging လုပ်ပါ
- Performance & Security ကိုဂျုဏိုက်ပါ
- Version Control & CI/CD ကိုအလေးထားပါ



Oracle Certifications

ORACLE®

Certified Master

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



Oracle Java Certifications

Oracle Java Certification ဆိတာက Oracle ကုမ္ပဏီက စစ်ဆေးပေးတဲ့ Java programming နည်းပညာရှင်တွေရဲ့ အရည်အချင်းကို အသိအမှတ်ပြုတဲ့ လက်မှတ်တစ်ခုဖြစ်ပါတယ်။ ဒီလက်မှတ်ရှိခိုင် Oracle ရဲ့ သတ်မှတ်ထားတဲ့ စာမေးပွဲတွေကို ဖြေဆိုအောင်မြင်ဖို့ လိုအပ်ပါတယ်။ Java developer တစ်ယောက်အနေနဲ့ သင့်ရဲ့ professional skills တွေကို သက်သေပြနိုင်မယ့် အခွင့်အရေးတစ်ခုလည်း ဖြစ်ပါတယ်။

□ Java Certification အမျိုးအစားများ

Oracle Java Certification တွေကို အဆင့်လိုက် ခွဲခြားထားပါတယ်။ အခြေခံကနေ အဆင့်မြင့်အထိ ရွှေးချယ်လို့ရပါတယ်။

- Oracle Certified Associate (OCA): Java ကို အခြေခံကစြိုးလေ့လာနေသူတွေအတွက် ရည်ရွယ်ပါတယ်။ ဥပမာ - Java SE 8 Programmer I
- Oracle Certified Professional (OCP): Intermediate နဲ့ Advanced level Java developer တွေအတွက် ဖြစ်ပါတယ်။ ဥပမာ - Java SE 11 Developer
- Oracle Certified Master (OCM): Java မှာ အဆင့်မြင့် ကျွမ်းကျင်သူတွေအတွက် ဖြစ်ပါတယ်။
- Oracle Certified Expert (OCE): Specialized areas ဖြစ်တဲ့ Web Services, EE နဲ့ ပတ်သက်တဲ့ certification တွေပါဝင်ပါတယ်။

□ Certification ရဲ့ အကျိုးကျေးဇူးများ

Java certification ရရှိခြင်းဖြင့် အောက်ပါ အကျိုးကျေးဇူးတွေကို ရရှိမှာဖြစ်ပါတယ်။

- ✓ အလုပ်အကိုင်အခွင့်အလမ်း ပိုမိုကောင်းမွန်လာမယ် - Certified Java developer တွေကို လစာပိုမိုကောင်းမွန်စွာ ပေးလေ့ရှိပါတယ်။
- ✓ Professional credibility တိုးတက်လာမယ် - သင့်ရဲ့ skills တွေကို ကဗျာနဲ့ချီးခြား အသိအမှတ်ပြုခဲ့ရမှာဖြစ်ပါတယ်။
- ✓ Java နဲ့ပတ်သက်တဲ့ နှုန်းကြိုင်းရှိုင်း အသိပညာတွေ ရရှိမယ် - Certification အတွက် လေ့လာရင်းနဲ့ Java ရဲ့ core concepts တွေကို ပိုမိုနားလည်လာမှာဖြစ်ပါတယ်။



□ စာမေးပွဲဖြန့်နည်း

Oracle Java certification စာမေးပွဲကို ဖြန့်ဖို့အတွက် အောက်ပါအဆင့်တွေကို လုပ်ဆောင်ရပါမယ်။

သင့်တော်တဲ့ certification level ကိုရွှေ့ချယ်ပါ - သင့်ရဲ့ current skill level ပေါ်မှုတည်ပြီး OCA, OCP (သို့) အခြား certification တစ်ခုခုကို ရွှေ့ချယ်နိုင်ပါတယ်။

လေ့လာပါ - Oracle ရဲ့ official study guides, online courses (Udemy, Coursera) နဲ့ practice exams တွေကို အသုံးပြုပြီး လေ့လာနိုင်ပါတယ်။

စာမေးပွဲအတွက် register လုပ်ပါ - Pearson VUE ကနေ online (သို့) test center မှာ ဖြန့်နိုင်ပါတယ်။
စာမေးပွဲဖြန့်ပါပဲ - Multiple-choice questions, coding exercises နဲ့ practical scenarios တွေပါဝင်နိုင်ပါတယ်။

□ ဘယ်လိုပြင်ဆင်ရမလဲ?

Java certification အောင်မြင်ဖို့အတွက် အောက်ပါ tips တွေကို အသုံးပြုနိုင်ပါတယ်။

- ✓ Official Java Documentation ကိုဖတ်ပါ - Oracle ရဲ့ Java docs စဲ certification အတွက် အရေးကြီးတဲ့ reference တစ်ခုဖြစ်ပါတယ်။
- ✓ Hands-on coding practice လုပ်ပါ - Real-world projects တွေမှာ Java ကို အသုံးပြုပြီး လေ့ကျင့်ပါ။
- ✓ Mock exams ဖြပ်ပါ - Enthuware, Whizlabs နဲ့ Udemy မှာ practice tests တွေရှိပါတယ်။

Oracle Java Certification ဟာ Java developer တစ်ယောက်အနေနဲ့ သင့်ရဲ့ career path ကို မြှင့်တင်ပေးနိုင်တဲ့ အရေးကြီးတဲ့ အခွင့်အလမ်းတစ်ခုဖြစ်ပါတယ်။ ဒါကြောင့် သင့်ရဲ့ Java skills တွေကို ပိုမိုတိုးတက်စေဖို့ certification တစ်ခုခုကို ယူထားသင့်ပါတယ်။



Online References for Self-study



- မြန်မာပြည်သားတိုင်း ကွန်ပျူးဘာ တတ်ရမည်။



ONLINE REFERENCES FOR SELF-STUDY

Java programming ကို self-study လုပ်ဖို့အတွက် online references အများကြီးရှိပါတယ်။ ဒါ references တွေက beginner ဘန် advanced level အထိ အဆင်ပြေပြေလေ့လာနိုင်ပါတယ်။ အောက်မှာ အသုံးဝင်မယ့် websites တွေနဲ့ resources တွေကို ဖော်ပြပေးထားပါတယ်။

Oracle Java Tutorials ၊ official Java documentation ဖြစ်ပြီး အခြေခံကနေ အဆင့်မြင့် Java concepts တွေကို ရှင်းပြထားပါတယ်။ ဒါ tutorial တွေက structured way နဲ့ ရေးထားတာမို့ step-by-step လိုက်လုပ်ရင်း Java ကို ကောင်းကောင်းနားလည်နိုင်မယ်။

link: Oracle Java Tutorials

W3Schools Java Tutorial ကလည်း beginner-friendly ဖြစ်ပြီး ရိုးရှင်းတဲ့ examples တွေနဲ့ Java basics ကို သင်ပေးပါတယ်။ interactive coding exercises တွေလည်းပါတာမို့ လက်တွေ့အလွယ်တကူလေ့ကျင့်နိုင်မယ်။

link: W3Schools Java

Codecademy မှာ Java course တွေရှိပြီး hands-on learning approach နဲ့ သင်ပေးပါတယ်။ project-based learning ဖြစ်တာမို့ real-world applications တွေအတွက် Java ကို ဘယ်လိုသုံးရမယ်ဆိုတာ နားလည်လာမယ်။

link: Codecademy Java

Udemy နဲ့ Coursera တို့မှာလည်း Java နဲ့ပတ်သက်တဲ့ paid/free courses တွေအများကြီးရှိပါတယ်။ ဒါ platforms တွေမှာ video lectures, assignments, နဲ့ quizzes တွေပါတာမို့ စနစ်တကျလေ့လာနိုင်မယ်။

YouTube မှာလည်း free Java tutorials တွေအများကြီးရှိပါတယ်။ Channels တွေဖြစ်တဲ့ Programming with Mosh, Bro Code, နဲ့ freeCodeCamp တို့က Java ကို အလွယ်တကူနားလည်အောင် ရှင်းပြထားပါတယ်။

GitHub နဲ့ Stack Overflow တို့ကိုလည်း Java နဲ့ပတ်သက်တဲ့ problem တွေဖြေရှင်းဖို့အတွက် အသုံးပြုနိုင်ပါတယ်။ Open-source projects တွေကို GitHub မှာလေ့လာပြီး coding skills တိုးတက်အောင် လုပ်နိုင်ပါတယ်။



Self-study လုပ်တဲ့အခါ practice များများလုပ်ဖို့အရေးကြီးပါတယ်။ Online coding platforms ဖြစ်တဲ့ Sololearn, HackerRank, LeetCode, နဲ့ CodeSignal တို့မှာ Java problems တွေကို လေ့ကျင့်နိုင်ပါတယ်။

ဒီ resources တွေကို အသုံးပြုပြီး Java programming ကို တစ်ဆင့်ချင်းလေ့လာနိုင်ပါတယ်။ စိတ်ရှည်ရှည်နဲ့ အချိန်ပေးပြီး လေ့ကျင့်မယ်ဆိုရင် Java ကို ကောင်းကောင်းတတ်မြောက်လာမှာဖြစ်ပါတယ်။

**■ References**

1. <https://www.w3schools.com/java/>
2. <https://www.tutorialspoint.com/java>
3. <https://freecodecamp.org>
4. <https://sololearn.com>
5. <https://openai.com/index/chatgpt>
6. <https://chat.deepseek.com>
7. Andrew Wellings. Concurrent and Real-Time Programming in Java. John Wiley & Sons, 2004.
8. Head First Java 3rd Edition 2023 by Sierra, Kathy Bates, Bert Gee
9. Joshua Bloch. Effective Java Programming Language Guide. Addison–Wesley, 2001

**■ ဓားရေးသူ၏ ကိုယ်ရေးအကျဉ်း**

ဤစာအုပ်ကိုရေးသာပြုစုသူကတော့ ကျွန်တော် ဆရာတင်မိုင်၏။ ကျွန်တော်က မြစ်ကြီးနားမြို့၊ အထက် (၁) မှာ အထက်တန်းကို ၁၉၉၄ မှာ အောင်မြင်ခဲ့ပါတယ်။ ပြီးတော့ ကျွန်တော် ဖိလိပိုင်နှင့် University of the Philippines (Los Banos) မှာ B.Sc Computer Science ကိုဆက်လက်ပညာသင်ယူခဲ့ပါတယ်။ ၂၀၀၄ မှာ B.Sc Computer Science နဲ့ဘွဲ့ရကျောင်းပြီးခဲ့ပါတယ်။ ၂၀၀၅ မှာ မြန်မာပြည်ပြန်လာပြီး မြစ်ကြီးနား အတိုမြို့မှာ Northern City Computer Training Center ကွန်ပျူးတာသင်တန်းကျောင်းကို စတင်တည်ထောင်ဖွံ့ဖွံ့ဖြစ်ခဲ့ပါတယ်။ လေ့လာဆည်းပူးခဲ့တဲ့ programming IT ဘာသာရပ်တွေကို သင်ကြားပို့ချခဲ့တာ ဖြစ်ပါတယ်။ ၂၀၀၉ မှာ မလေးရှားနိုင်ငံ Kualalumpur မှာ Zepto IT Solution လိုပေါ်တဲ့ အိုင်တီ Company မှာ Software Developer (programmer) အဖြစ် ၃ နှစ်တာ အလုပ်လုပ်ခဲ့ပါတယ်။ ကျွန်းမာရေးအဖော်နေဂြာင်း ရန်ကုန်မြို့ကို ပြန်လာပြီး ဆေးကုသရင်း ရန်ကုန်မြို့မှာပဲ ValueStar Computer Institute မှ ၆ နှစ်တာ IT Lecturer အနေဖြင့် IT Department Head အဖော်ရော့ ဆက်လက်ခြေချခဲ့ပါတယ်။ ၂၀၁၇ မှာ ကိုယ်ပိုင် အိုင်တီသင်တန်းကျောင်းအဖြစ် Northern City Center နာမည်နဲ့ အရင်က မြစ်ကြီးနားမှာ တည်ထောင်ခဲ့ဖူး တဲ့ နာမည်ကို ပြန်လည်အသုံးပြုကာ ဖွင့်လှစ်ထားပြီး ၂၀၂၅ လက်ရှုအချိန်အထိ သင်တန်းကျောင်းကို ဦးစီးလုပ်ကိုင်နေဆဲ ဖြစ်ပါတယ်။

- မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။**

**■ စာရေးသူ၏ အိုင်တိအတွေးအကြံ မှတ်တမ်းများ**

Popular Web projects –

- Japan Used Car Sales & Show room
<https://www.sbtjapan.com/sbt-myanmar/>
- Myanmar Traditional Boxing
<https://www.myanmartraditionalboxing.com.mm>
- Myanmar Agriculture Machinery & Products
<https://ptyeeshinn.com.mm>
- Real Estate
<https://www.saikhungnoung.com>
- Malaysia Minimart
<https://familystore.com.my>
- China Products & Fashion Sales
<https://youhome.space/>
- Online Payment System
<https://ucapay.com.mm>

Popular Point of Sales Application Projects –

- Malaysia Family Store Desktop Application and Mobile Application
- Myitkyina iGu Café & Gallery Desktop Application
- Myitkyina Hospital Blood Bank Desktop Application
- Myitkyina Fuji Food & Drinks Desktop Application
- Yangon Joyzone Stock Controller Desktop Application and Mobile Application
- Yangon Malihka Restaurant Desktop Application and Mobile Application
- Yangon Yuri Café Mobile Application and Mobile Application
- Yangon Gracevines Agarwood Desktop Application and Mobile Application



■ Documentary Photos



■ မြန်မာပြည်သားတိုင်း ကွန်ပျူးတာ တတ်ရမည်။

