Zoo Management System Documentation

Package and Interface
The code is organized into a package named org.example. Inside this package, there is an interface named Zood, which defines a set of methods that must be implemented by classes that implement this interface.

Zood Interface
The Zood interface provides a blueprint for a set of functionalities that must be implemented by classes managing a zoo. These functionalities include:

addAttraction(String name, String description, int price, int count): Adds an attraction to the zoo with the specified name, description, price, and count.
viewAttractions(): Displays information about all the attractions in the zoo.
getAttractions(): Returns a list of attractions in the zoo.
addAnimal(String name, String type): Adds an animal to the zoo with the specified name and type.
viewAnimals(): Displays information about all the animals in the zoo.
getAnimals(): Returns a list of animals in the zoo.
addDiscount(String category, double percent): Adds a discount for a specified category.
getDiscounts(): Returns a map of discount categories and their associated percentages.
registerVisitor(String name, int age, String phoneNumber, double balance, String email, password): Registers a new visitor with the provided information.
addFeedback(String feedback): Adds feedback to the list of feedback received by the zoo.
getFeedback(): Displays the feedback received by the zoo.
Zoo Class
The Zoo class implements the Zood interface and represents a zoo. This class has several fields, a constructor, and methods to manage various aspects of the zoo.

Fields
attractions: A list of attractions in the zoo.
animals: A list of animals in the zoo.
discounts: A map that stores discount categories and their associated percentages.
feedbacks: A list of feedback received by the zoo.
visitors: A list of registered visitors to the zoo.
Constructor
The constructor initializes the animals list with some default animals and sets up other data structures.

Methods
The Zoo class provides various methods to manage the zoo:

addAttraction(String name, String description, int price, int count): Adds a new attraction to the zoo.
viewAttractions(): Displays information about all the attractions in the zoo.
getAttractions(): Returns a list of attractions in the zoo.
addAnimal(String name, String type): Adds a new animal to the zoo.
viewAnimals(): Displays information about all the animals in the zoo.
getAnimals(): Returns a list of animals in the zoo.
addDiscount(String category, double percent): Adds a discount for a specified category.
getDiscounts(): Returns a map of discount categories and their associated percentages.
getVisitor(String username, String password): Returns a visitor with the given username and password.
addFeedback(String feedback): Adds feedback to the list of feedback received by the zoo.
getFeedback(): Displays the feedback received by the zoo.
Visitor Class (Inner Class)
The Visitor class represents a visitor to the zoo and is an inner class within the Zoo class. It includes fields and methods related to individual visitors.

Fields
name: Name of the visitor.
age: Age of the visitor.
phoneNumber: Phone number of the visitor.
balance: Visitor's balance.
email: Visitor's email address.
password: Visitor's password.
Constructor
The constructor initializes the visitor with the provided information.

Methods
setBalance(double balance): Sets the visitor's balance to the given value.
Attractions Class (Abstract Class)
The Attractions class represents a general concept of attractions in the zoo. This is an abstract class that serves as a base for specific attractions.

Fields
name: Name of the attraction.
description: Description of the attraction.
price: Price of the attraction.
count: Visitor count for the attraction.
Methods
getName(): Returns the name of the attraction.
getDescription(): Returns the description of the attraction.
setCount(): An abstract method to set the count of visitors for the attraction.
getVisitorCountAttraction(): Returns the visitor count for the attraction.
Attraction Class

The Attraction class represents a specific attraction in the zoo. It is a concrete class that inherits from the Attractions class.

Fields
name: Name of the attraction.
description: Description of the attraction.
price: Price of the attraction.
count: Visitor count for the attraction.
Constructor
The constructor initializes the attraction with the provided information.

Methods
Overrides setCount() to increment the visitor count.
Overrides getVisitorCountAttraction() to return the visitor count.
Entity Class (Abstract Class)
The Entity class represents a general concept of an entity with a name. This is an abstract class that serves as a base for specific entities.

Fields
name: Name of the entity.
Methods
getName(): Returns the name of the entity.
Admin Class
The Admin class represents an admin entity. It includes fields and methods related to administrators of the zoo.

Fields
password: Password of the admin.
Constructor
The constructor initializes the admin with the provided username and password.

Methods
getPassword(): Returns the admin's password.
Main Class
The Main class is the main class that contains the main method, serving as the entry point for the program.

Fields
scanner: A Scanner object used to read user input.
x: A variable used to control the main loop.
adminLoggedIn: A boolean flag indicating whether an admin is logged in.
visitorLoggedIn: A boolean flag indicating whether a visitor is logged in.
zoo: An instance of the Zoo class, representing the zoo.
deal: A string variable to store special deals.

Main Method

The main method is the entry point of the program. It provides a menu-driven interface that allows users to interact with the zoo management system. The main functionalities of the main method include:

Displaying a welcome message.
Allowing users to log in as an admin or a visitor, or view special deals.
Handling admin interactions, including managing attractions, animals, discounts, special deals, visitor statistics, and feedback.
Handling visitor interactions, including registration, login, and various visitor-specific actions such as exploring the zoo, buying memberships and tickets, viewing discounts and special deals, visiting animals and attractions, leaving feedback, and logging out.
Implementing error handling to ensure valid user input.

Error Handling

Error handling is an essential part of the program. It ensures that user inputs are validated and provides feedback for any invalid inputs. This includes checking for valid choices, data types, and unique entries (such as usernames, phone numbers, and email addresses).

Exiting the Program

The program continues to run until the user explicitly chooses to exit. This is achieved by providing appropriate menu options, including options for logging out or exiting the program.

Encapsulation & Class Relationships

The code demonstrates the concept of encapsulation by using classes to encapsulate related data and behaviors. Each class, including Zoo, Visitor, Attraction, Entity, and Admin, encapsulates relevant fields and methods related to their respective entities. This ensures that data and operations are contained within the appropriate classes.

Class relationships are also evident in the code:

The Zoo class implements the Zood interface, establishing a relationship between the interface and the implementing class.
The Attraction class extends the Attractions abstract class, demonstrating an inheritance relationship.
The Visitor class is an inner class within the Zoo class, illustrating a containment relationship.
The Admin class is a separate entity with its own fields and methods for admin-related functionality.
These class relationships enable the code to model the zoo management system effectively and maintain a clear and organized structure.

Error Handling

User Input Validation

Error handling is implemented in the code to ensure that user input is valid. It plays a crucial role in making the program robust and user-friendly. Here's how error handling is done:

Choice Validation: The code includes loops to validate user choices. When the user is prompted to enter a choice, the program checks if the input is a valid integer using if (scanner.hasNextInt()). If the input is not a valid integer, the program provides an error message and continues to request input until a valid choice is entered.

Data Type Validation: For various user inputs such as age, balance, price, and discount percentage, the code uses hasNextInt() and hasNextDouble() to check if the entered data is of the correct data type. If the input is incorrect, the program displays an error message and prompts the user to enter the correct data type.

Unique Entries: When registering a visitor, the code ensures that usernames, phone numbers, and email addresses are unique. It maintains sets (uniquePhoneNumbers, uniqueEmails, uniqueUsernames) to store and validate uniqueness. If a duplicate entry is detected, the program provides an error message and prompts the user to enter unique information.

Invalid Choices: In various parts of the program, the user is prompted to make choices (e.g., explore atREADMEtractions or animals, buy memberships, etc.). If the user enters an invalid choice, the program informs them about the error and continues to request a valid choice.

Discount Code Validation: When purchasing a membership or a ticket, the code checks if the entered discount code is valid. If it's a recognized code, a corresponding discount is applied. If the code is not recognized, an error message is displayed.

User Feedback
In case of invalid input or errors, the code provides user-friendly error messages, such as "Invalid input. Please enter a valid integer" or "Username already in use. Please enter a unique username." This helps users understand why their input was rejected and guides them toward providing correct input.

Class Relationships
Interface Implementation
In the code, the Zoo class implements the Zood interface. This demonstrates a relationship between the interface and the implementing class. The Zood interface defines a set of methods that the Zoo class must implement. This relationship ensures that the Zoo class adheres to the contract defined by the interface, enabling it to provide the required functionalities.

Inheritance
The code includes a class hierarchy where the Attraction class extends the Attractions abstract class. This represents an inheritance relationship. The Attractions class serves as an abstract base class for attractions in the zoo. The Attraction class inherits the properties and behaviors

defined in the Attractions class, allowing it to provide concrete implementations of methods and properties. This relationship promotes code reusability and abstraction.

Inner Class
The code features an inner class named Visitor within the Zoo class. An inner class is a class defined within another class, and it has access to the enclosing class's members. In this case, the Visitor class encapsulates visitor-specific attributes and behavior within the Zoo class, forming a containment relationship. This relationship allows the Visitor class to access and interact with the data and methods of the outer Zoo class. For example, it can modify the balance of a visitor directly.

Conclusion
Error handling is critical for validating user input and providing feedback when errors occur. The code demonstrates effective error handling techniques by checking for valid data types, unique entries, and invalid choices while interacting with the program. Additionally, the code exhibits different types of class relationships, including interface implementation, inheritance, and inner class containment. These relationships help maintain code structure, reusability, and encapsulation.

Admin Login Credentials:

The admin can log in with the username "admin" and password "admin123". These are predefined admin login credentials.

Visitor Phone Number Format:
The format for a visitor's phone number is not strictly enforced. It can be any string, including numbers and special characters. For example, "0129-123456" or "(555) 555-5555" are acceptable formats.
Phone Number is Unique

Visitor Email Address Format:
The format for a visitor's email address is not strictly enforced. It can be any string with the "@" symbol, such as "user123@gmail.com" or "visitor@example.com". There are no specific email address validation rules.
Email Should be unique

Visitor Username
This is Unique

Automatically Discount
Automatic Discount of 10% for age below 18 and 20 for age above 65

Special Deals:

The system allows for special deals, but the details of these deals and how they are created or applied are not specified .

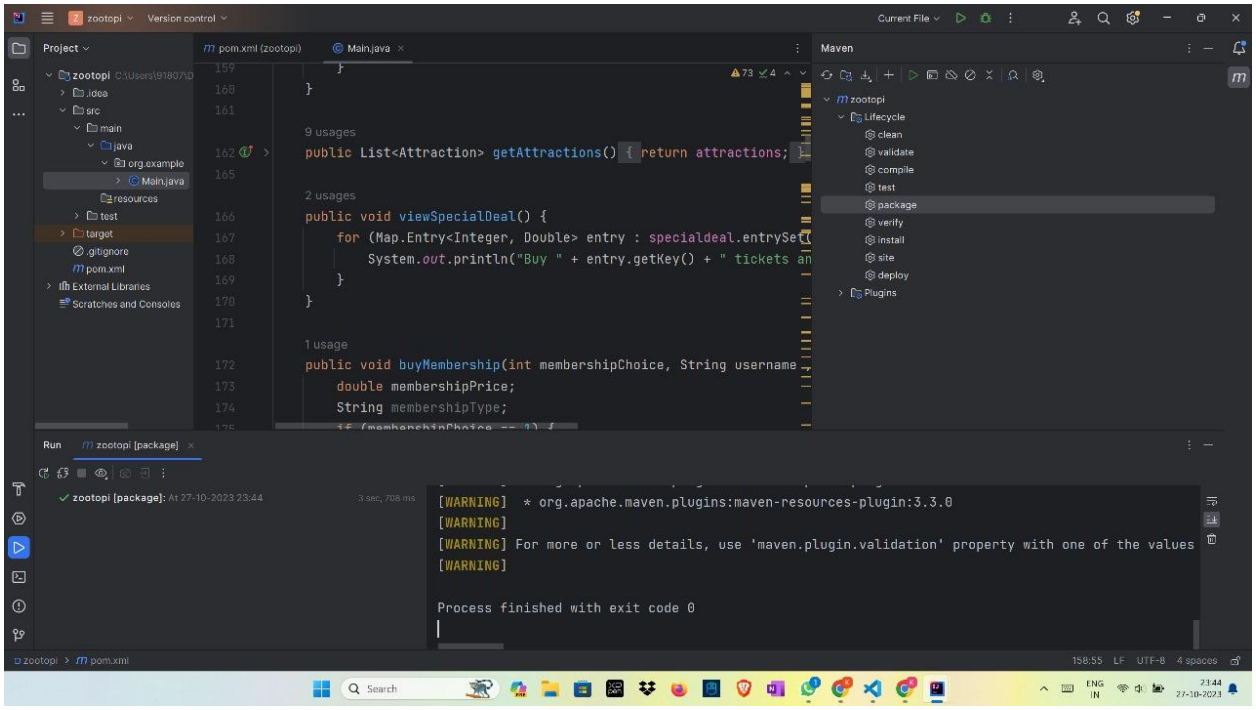
Visitor Feedback:
Visitors can leave feedback, but the format, length, or specific categories for feedback are not defined.

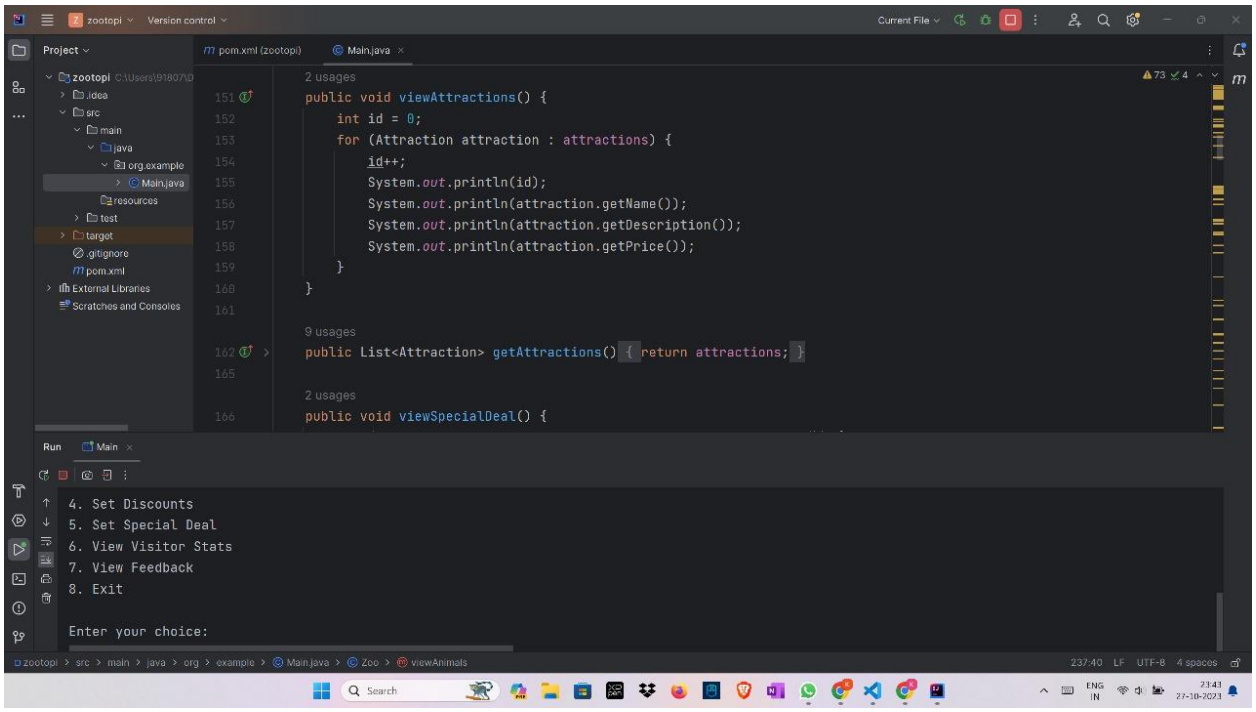
Attraction Schedule Status:
Attractions have schedule status, which can be set to various values, including "CLOSED." However, the exact schedule statuses and their meanings are not detailed .

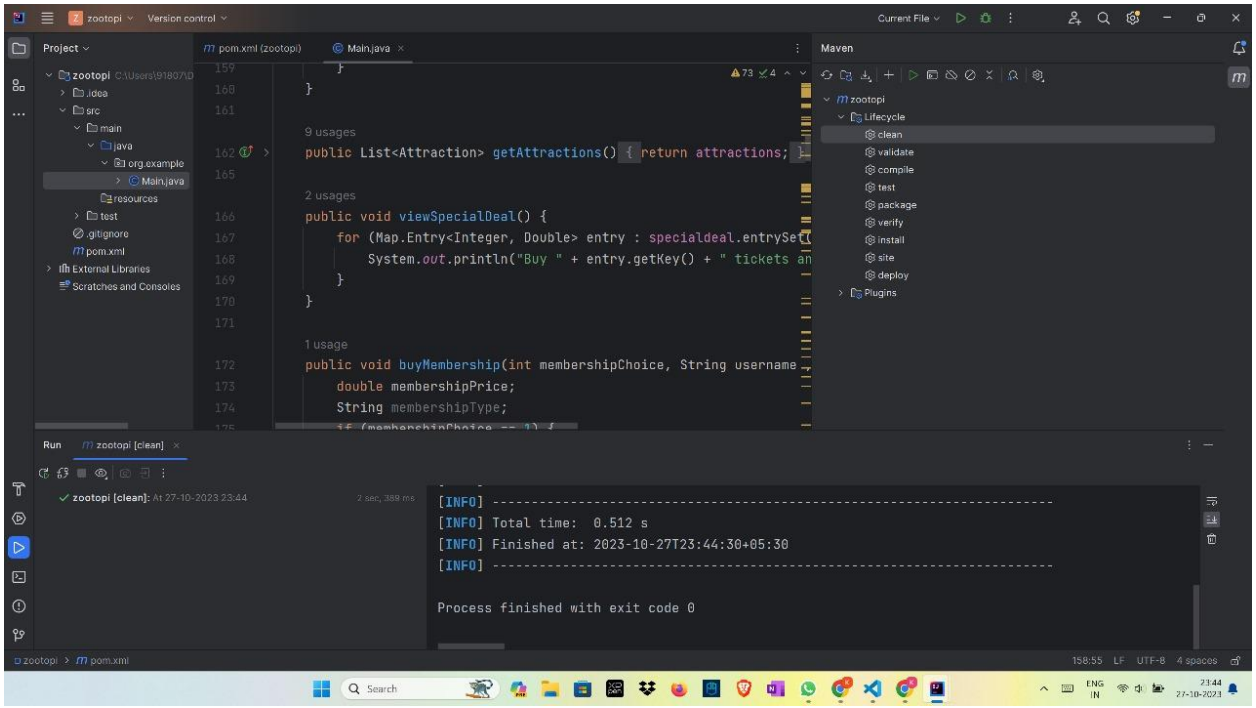
Discounts:
The system supports discounts for minors and seniors. The exact discount percentages and how they are applied are not defined.

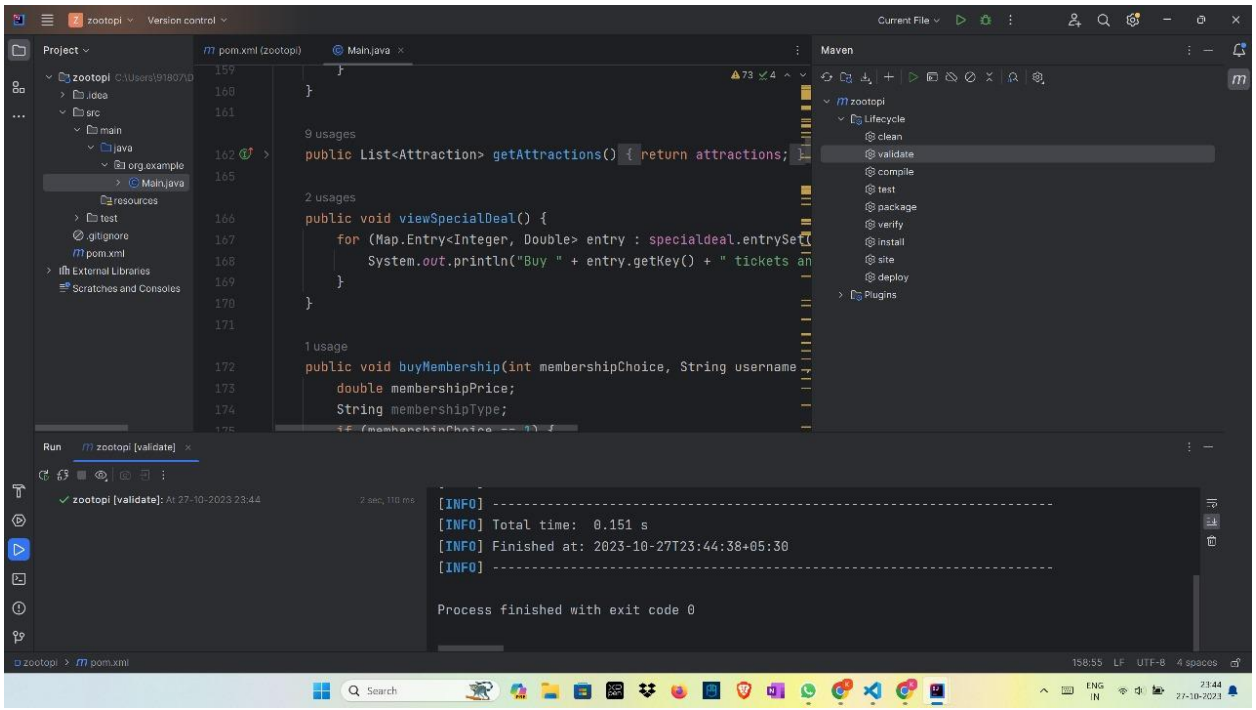
Account Balance:
Visitors have an account balance, but the initial balance, currency, or how it is replenished are not specified.

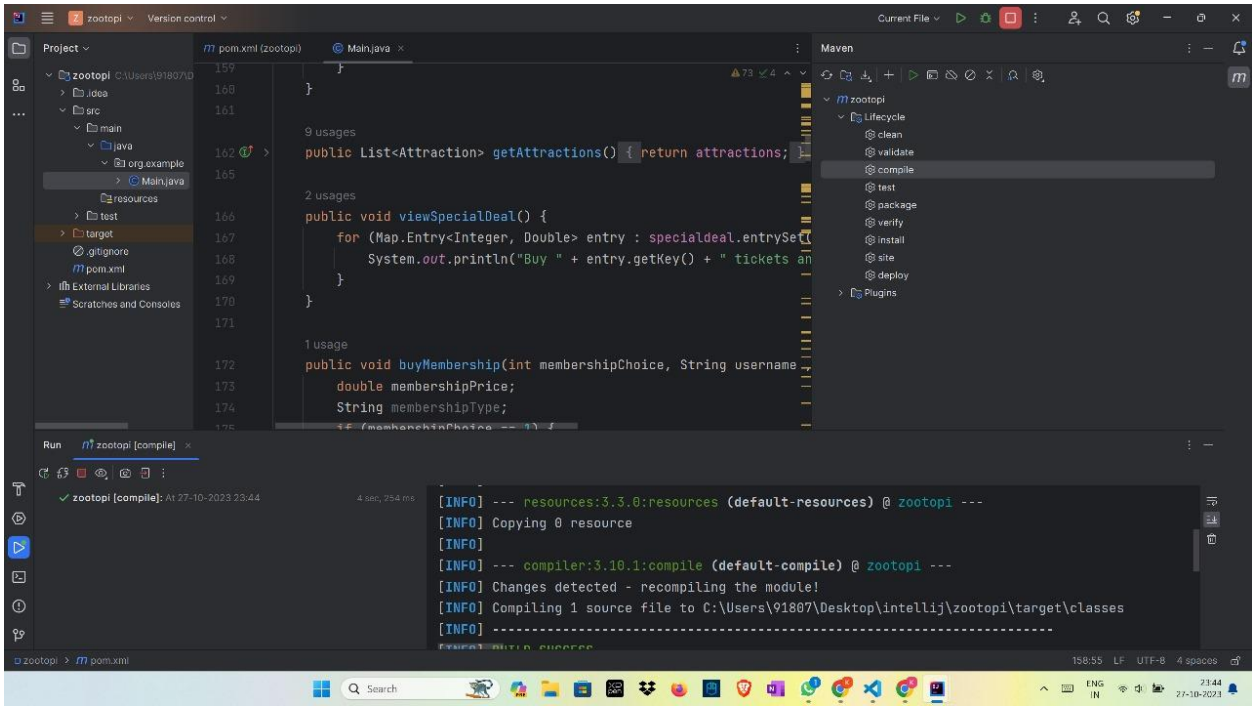
Visitor Spending:
The system keeps track of a visitor's spending, but the currency or specific items related to spending are not detailed.

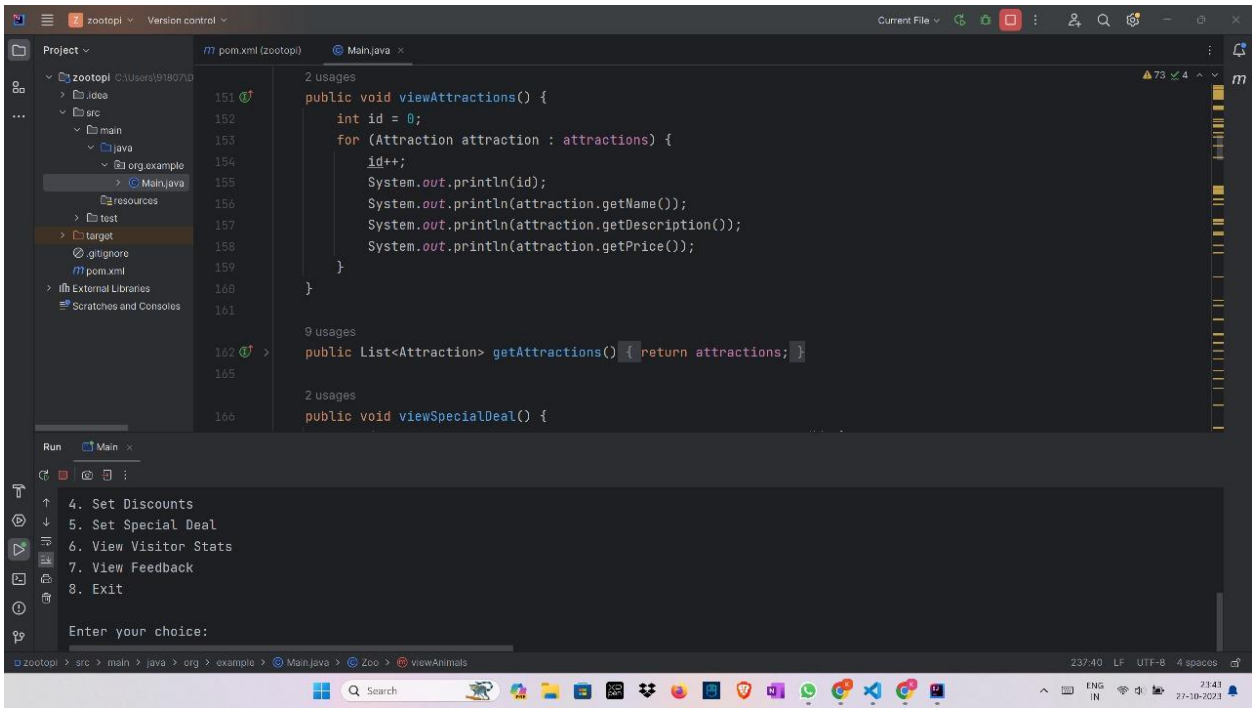

zootopi
Version control
Current File
Project
pom.xml (zootopi)
Main.java
Maven
zootopi
Lifecycle
clean
validate
compile
test
package
verify
install
site
deploy
Plugins
9 usages
public List<Attraction> getAttractions() { return attractions; }
2 usages
public void viewSpecialDeal() {
for (Map.Entry<Integer, Double> entry : specialdeal.entrySet
System.out.println("Buy " + entry.getKey() + " tickets an
}
}
1 usage
public void buyMembership(int membershipChoice, String username
double membershipPrice;
String membershipType;
Run
zootopi [package]
zootopi [package]: At 27-10-2023 23:44
3 sec, 708 ms
[WARNING] * org.apache.maven.plugins:maven-resources-plugin:3.3.0
[WARNING]
[WARNING] For more or less details, use 'maven.plugin.validation' property with one of the values
[WARNING]
Process finished with exit code 0
158:55
LF
UTF-8
4 spaces
Search
ENG
IN
23:44
27-10-2023


zootopi
Version control
Current File
Project
pom.xml (zootopi)
Main.java
2 usages
public void viewAttractions() {
int id = 0;
for (Attraction attraction : attractions) {
id++;
System.out.println(id);
System.out.println(attraction.getName());
System.out.println(attraction.getDescription());
System.out.println(attraction.getPrice());
}
}
9 usages
public List<Attraction> getAttractions() { return attractions; }
2 usages
public void viewSpecialDeal() {
Run
Main
4. Set Discounts
5. Set Special Deal
6. View Visitor Stats
7. View Feedback
8. Exit
Enter your choice:
zootopi > src > main > java > org > example > Main.java > Zoo > viewAnimals
237:40
LF
UTF-8
4 spaces
Search
ENG
IN
23:41
27-10-2023


zootopi
Version control
Current File
Project
pom.xml (zootopi)
Main.java
Maven
zootopi
Lifecycle
clean
validate
compile
test
package
verify
install
site
deploy
Plugins
9 usages
public List<Attraction> getAttractions() { return attractions; }
2 usages
public void viewSpecialDeal() {
for (Map.Entry<Integer, Double> entry : specialdeal.entrySet(
System.out.println("Buy " + entry.getKey() + " tickets an
}
}
1 usage
public void buyMembership(int membershipChoice, String username
double membershipPrice;
String membershipType;
Run
zootopi [clean]
zootopi [clean]: At 27-10-2023 23:44
2 sec, 389 ms
[INFO] --
[INFO] Total time: 0.512 s
[INFO] Finished at: 2023-10-27T23:44:30+05:30
[INFO] --
Process finished with exit code 0
158:55
LF
UTF-8
4 spaces
Search
ENG
IN
23:44
27-10-2023


zootopi
Version control
Current File
Project
pom.xml (zootopi)
Main.java
Maven
zootopi
Lifecycle
clean
validate
compile
test
package
verify
install
site
deploy
Plugins
9 usages
public List<Attraction> getAttractions() { return attractions; }
2 usages
public void viewSpecialDeal() {
for (Map.Entry<Integer, Double> entry : specialdeal.entrySet(
System.out.println("Buy " + entry.getKey() + " tickets an
}
}
1 usage
public void buyMembership(int membershipChoice, String username
double membershipPrice;
String membershipType;
Run
zootopi [validate]
zootopi [validate]: At 27-10-2023 23:44
2 sec, 110 ms
[INFO] --
[INFO] Total time: 0.151 s
[INFO] Finished at: 2023-10-27T23:44:38+05:30
[INFO] --
Process finished with exit code 0
158:55
LF
UTF-8
4 spaces
Search
ENG
IN
23:44
27-10-2023


zootopi
Version control
Current File
Project
pom.xml (zootopi)
Main.java
Maven
zootopi
Lifecycle
clean
validate
compile
test
package
verify
install
site
deploy
Plugins
9 usages
public List<Attraction> getAttractions() { return attractions; }
2 usages
public void viewSpecialDeal() {
for (Map.Entry<Integer, Double> entry : specialdeal.entrySet(
System.out.println("Buy " + entry.getKey() + " tickets an
}
}
1 usage
public void buyMembership(int membershipChoice, String username
double membershipPrice;
String membershipType;
Run
zootopi [compile]
zootopi [compile]: At 27-10-2023 23:44
4 sec, 254 ms
[INFO] --- resources:3.3.0:resources (default-resources) @ zootopi ---
[INFO] Copying 0 resource
[INFO]
[INFO] --- compiler:3.10.1:compile (default-compile) @ zootopi ---
[INFO] Changes detected - recompiling the module!
[INFO] Compiling 1 source file to C:\Users\91807\Desktop\intellij\zootopi\target\classes
[INFO] --
158:55
LF
UTF-8
4 spaces
Search
ENG
IN
23:44
27-10-2023


zootopi
Version control
Current File
Project
pom.xml (zootopi)
Main.java
2 usages
public void viewAttractions() {
int id = 0;
for (Attraction attraction : attractions) {
id++;
System.out.println(id);
System.out.println(attraction.getName());
System.out.println(attraction.getDescription());
System.out.println(attraction.getPrice());
}
}
9 usages
public List<Attraction> getAttractions() { return attractions; }
2 usages
public void viewSpecialDeal() {
Run
Main
4. Set Discounts
5. Set Special Deal
6. View Visitor Stats
7. View Feedback
8. Exit
Enter your choice:
237:40
LF
UTF-8
4 spaces
Search
ENG
IN
23:41
27-10-2023