Chapter 1: Introduction to Java

What is Java? and Its Importance

Java is a high-level, object-oriented programming language used for building platform-independent applications.

First Java Program

This code prints “Hello, World!” to the console, demonstrating basic Java syntax and structure.

```
public class HelloWorld {
public static void main(String[] args) {
System.out.println("Hello, World!");
}
}
```

Variables and Data Types

Variables store data of different types: int, double, String, and boolean.

```
int age = 25;
double salary = 55000.50;
String name = "Alice";
boolean isEmployed = true;
```

Constants

Constants are declared with `final` and cannot be changed after initialization.

```
final double PI = 3.14159;
```

Operations in Java

Basic arithmetic operations: addition, subtraction, multiplication, and division.

```
int a = 10;
int b = 5;
int sum = a + b;
int diff = a - b;
int prod = a * b;
int quot = a / b;
```

Chapter 2: Control Structures

Conditional Statements

Control the flow of execution with if-else-if statements.

```
int x = 10;
if (x > 0) {
System.out.println("Positive");
} else if (x == 0) {
System.out.println("Zero");
} else {
System.out.println("Negative");
}
```

Looping Constructs

For loop iterates over a range, while loop repeats as long as a condition is true.

```
// For loop
for (int i = 0; i < 5; i++) {
System.out.println(i);
}
// While loop
int i = 0;
while (i < 5) {
System.out.println(i);
i++;
}
```

Jump Statements

Break exits the loop, continue skips the current iteration.

```
// Break statement
for (int i = 0; i < 5; i++) {
if (i == 3) {
break;
```

```
}
System.out.println(i);
}
// Continue statement
for (int i = 0; i < 5; i++) {
if (i == 3) {
continue;
}
System.out.println(i);
}
```

Chapter 3: Methods

Defining Methods

Define methods using `public static` keywords with parameters and return type.

```
public class Main {
public static int add(int a, int b) {
return a + b;
}
public static void main(String[] args) {
System.out.println(add(2, 3));
}
}
```

Method Overloading

Method overloading allows multiple methods with the same name but different parameters.

```
public class Main {
public static int add(int a, int b) {
return a + b;
}
public static double add(double a, double b) {
return a + b;
}
public static void main(String[] args) {
System.out.println(add(2, 3));
System.out.println(add(2.5, 3.5));
}
}
```

Recursion

Recursion is a method calling itself to solve a problem by breaking it down into smaller problems.

```
public class Main {
```

```
public static int factorial(int n) {
if (n == 0) {
return 1;
} else {
return n * factorial(n - 1);
}
}
public static void main(String[] args) {
System.out.println(factorial(5));
}
}
```

Chapter 4: Object-Oriented Programming

Classes and Objects

Classes define objects with attributes and methods; instantiate objects using the class.

```
class Dog {
String name;
Dog(String name) {
this.name = name;
}
void bark() {
System.out.println(name + " says woof!");
}
}
public class Main {
public static void main(String[] args) {
Dog dog = new Dog("Buddy");
dog.bark();
}
}
```

Inheritance

Inheritance allows a class to inherit attributes and methods from another class.

```
class Animal {
void eat() {
System.out.println("This animal eats food.");
}
}
class Dog extends Animal {
void bark() {
System.out.println("This dog barks.");
}
}
```

```
public class Main {
public static void main(String[] args) {
Dog dog = new Dog();
dog.eat();
dog.bark();
}
}
```

Polymorphism

Polymorphism allows a method to do different things based on the object it is acting upon.

```
class Animal {
void sound() {
System.out.println("Animal makes a sound");
}
}
class Dog extends Animal {
void sound() {
System.out.println("Dog barks");
}
}
public class Main {
public static void main(String[] args) {
Animal myDog = new Dog();
myDog.sound();
}
}
```

Chapter 5: Exception Handling

Understanding Exceptions

Handle runtime errors using try-catch blocks.

```
public class Main {
public static void main(String[] args) {
try {
int result = 10 / 0;
} catch (ArithmeticException e) {
System.out.println("Cannot divide by zero");
}
}
}
```

Creating Custom Exceptions

Define custom exceptions by extending the Exception class.

```
class CustomException extends Exception {
CustomException(String message) {
super(message);
}
}
public class Main {
public static void main(String[] args) {
try {
throw new CustomException("Custom error occurred");
} catch (CustomException e) {
System.out.println(e.getMessage());
}
}
}
```

Chapter 6: Collections

Lists

Lists store ordered collections of elements, allowing duplicates.

```
import java.util.ArrayList;
import java.util.List;
public class Main {
public static void main(String[] args) {
List<Integer> numbers = new ArrayList<>();
numbers.add(1);
numbers.add(2);
numbers.add(3);
System.out.println(numbers);
}
}
```

Sets

Sets store unique elements, preventing duplicates.

```
import java.util.HashSet;
import java.util.Set;
public class Main {
public static void main(String[] args) {
Set<Integer> uniqueNumbers = new HashSet<>();
uniqueNumbers.add(1);
uniqueNumbers.add(2);
uniqueNumbers.add(2); // Duplicate
System.out.println(uniqueNumbers);
}
}
```

Maps

Maps store key-value pairs, allowing fast lookup by key.

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

```
import java.util.Map;
public class Main {
public static void main(String[] args) {
Map<String, Integer> ages = new HashMap<>();
ages.put("Alice", 25);
ages.put("Bob", 30);
System.out.println(ages);
}
}
```

Chapter 7: Lambda Expressions

Using Lambda Expressions

Lambda expressions provide a concise way to represent anonymous functions.

```
import java.util.Arrays;
import java.util.List;
public class Main {
public static void main(String[] args) {
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
names.forEach(name -> System.out.println(name));
}
}
```

Chapter 8: Streams

Using Streams

Streams provide a way to process sequences of elements in a functional style.

```
import java.util.Arrays;
import java.util.List;
public class Main {
public static void main(String[] args) {
List<String> names = Arrays.asList("Alice", "Bob", "Charlie");
names.stream()
.filter(name -> name.startsWith("A"))
.forEach(System.out::println);
}
}
```

Chapter 9: Date and Time

Working with Date and Time

Java provides the `java.time` package for date and time manipulation.

```
import java.time.LocalDate;
import java.time.LocalTime;
import java.time.LocalDateTime;
public class Main {
public static void main(String[] args) {
LocalDate date = LocalDate.now();
LocalTime time = LocalTime.now();
LocalDateTime dateTime = LocalDateTime.now();
System.out.println("Date: " + date);
System.out.println("Time: " + time);
System.out.println("DateTime: " + dateTime);
}
}
```

Chapter 10: Regular Expressions

Using Regular Expressions

Regular expressions are used for pattern matching within strings.

```
import java.util.regex.Pattern;
import java.util.regex.Matcher;
public class Main {
public static void main(String[] args) {
String text = "Hello, World!";
String patternString = "\\bWorld\\b";
Pattern pattern = Pattern.compile(patternString);
Matcher matcher = pattern.matcher(text);
if (matcher.find()) {
System.out.println("Found match: " + matcher.group());
}
}
}
```