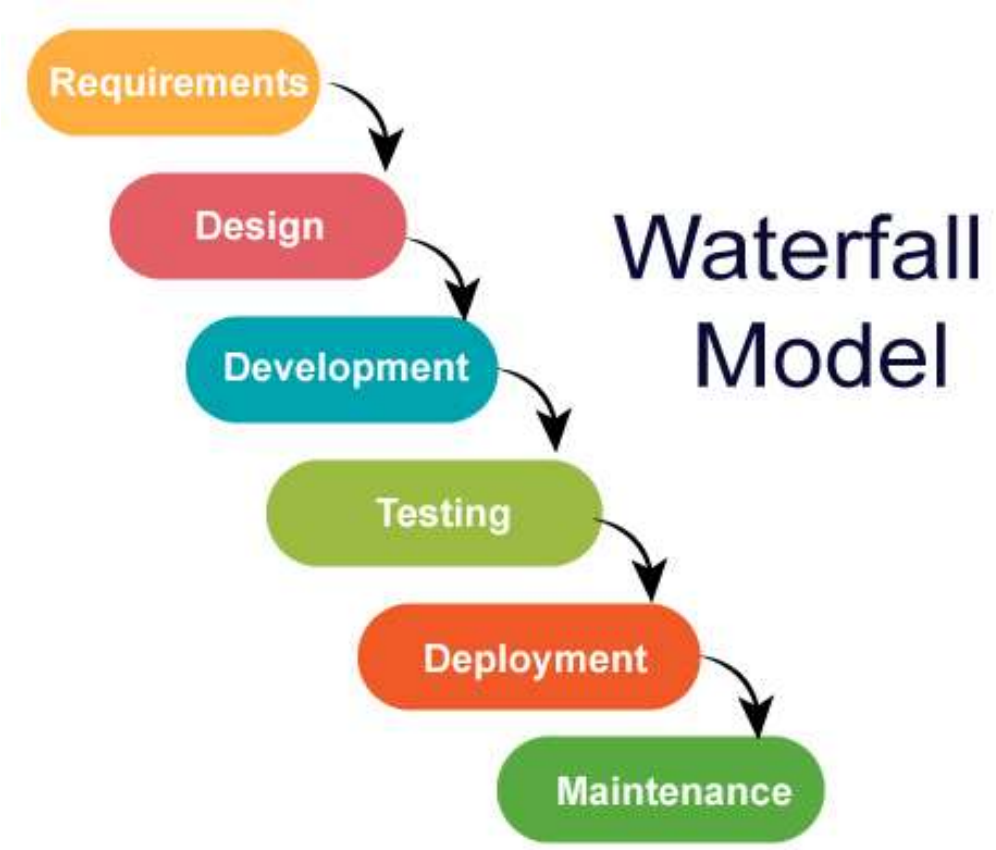
Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) consists of various models that guide the process of developing software. Here's a brief overview of some popular SDLC models:

1. Waterfall Model:
   - Linear and sequential model.
   - Progresses through phases like requirements, design, implementation, testing, deployment, and maintenance.



The Waterfall Model is a linear and sequential approach to software development. It consists of distinct phases that must be completed in a specific order, with each phase building on the results of the previous one. Here's a simple Java program that reflects the linear nature of the Waterfall Model

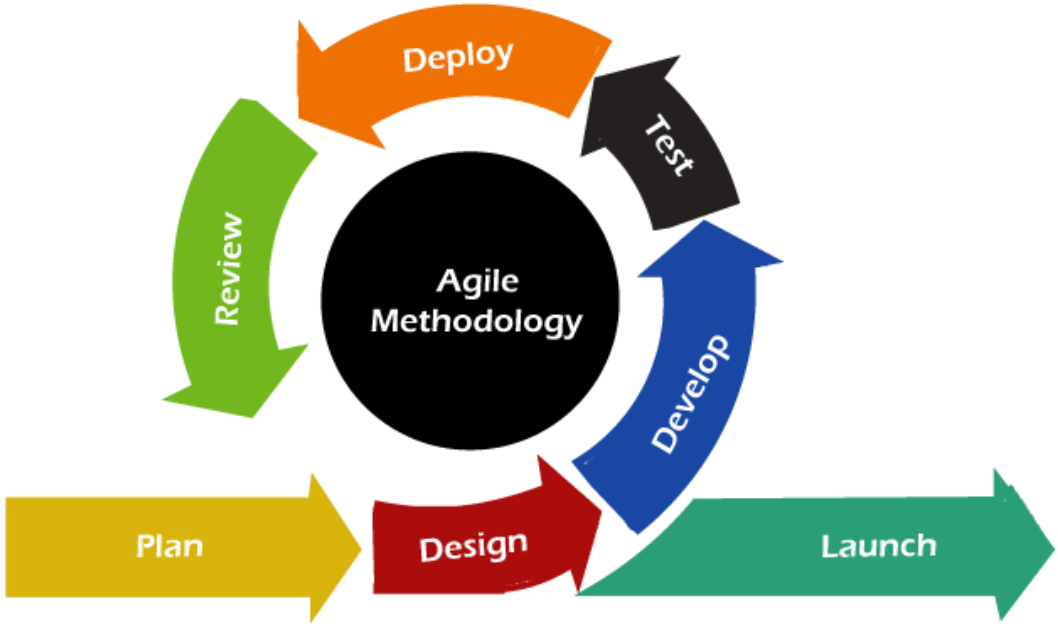
```java
import java.util.Scanner;

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

        // Requirements phase
        System.out.println("Waterfall Model: Requirements
Phase");
        System.out.println("Gather and document system
requirements.");

        // Design phase
        System.out.println("\nWaterfall Model: Design Phase");
        System.out.println("Create a detailed design based on
the requirements.");

        // Implementation phase
        System.out.println("\nWaterfall Model: Implementation
Phase");
        System.out.println("Code the software based on the
detailed design.");

        // Testing phase
        System.out.println("\nWaterfall Model: Testing
Phase");
        System.out.println("Perform testing to ensure the
software meets requirements.");

        // Deployment phase
        System.out.println("\nWaterfall Model: Deployment
Phase");
        System.out.println("Deploy the software for use.");

        // Maintenance phase
        System.out.println("\nWaterfall Model: Maintenance
Phase");
        System.out.println("Provide ongoing support and make
necessary updates.");

        System.out.println("\nWaterfall Model completed
successfully!");

        scanner.close();
    }
}
```

In this example, each phase of the Waterfall Model is represented, and the program outputs messages simulating the tasks performed in each phase. Note that in a real-world

scenario, each phase would involve more detailed and specific tasks, including documentation, collaboration with stakeholders, and more comprehensive testing procedures.

2. **Agile Model:**

- Emphasizes iterative and incremental development.
- Responds to changes quickly and allows flexibility.
- Involves collaboration between cross-functional teams.



The Agile Model is an iterative and incremental approach to software development, emphasizing flexibility and responsiveness to change. It involves the collaboration of cross-functional teams and aims to deliver small, working increments of software at regular intervals. Here's a simple Java program to illustrate the Agile Model concept using a basic user story and a sprint:

```java
import java.util.Scanner;

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

        // Agile Model: Sprint 1
        System.out.println("Agile Model: Sprint 1");
        System.out.println("User Story: As a user, I
want to log in to the system.");

        // Implementing user story for Sprint 1
        System.out.println("\nImplementing User
Story: Logging In");
        // Code for implementing user login
functionality

        // Testing user story for Sprint 1
        System.out.println("Testing User Story:
Logging In");
        // Code for testing user login functionality

        // Agile Model: Sprint 2
        System.out.println("\nAgile Model: Sprint
2");
        System.out.println("User Story: As a user, I
want to add items to my shopping cart.");

        // Implementing user story for Sprint 2
        System.out.println("\nImplementing User
Story: Adding Items to Cart");
        // Code for implementing shopping cart
functionality

        // Testing user story for Sprint 2
        System.out.println("Testing User Story:
Adding Items to Cart");
        // Code for testing shopping cart
functionality

        // Agile Model: Sprint 3
        System.out.println("\nAgile Model: Sprint
3");
```

```
        System.out.println("User Story: As a user, I
want to complete my purchase.");

        // Implementing user story for Sprint 3
        System.out.println("\nImplementing User
Story: Completing Purchase");
        // Code for implementing purchase
functionality

        // Testing user story for Sprint 3
        System.out.println("Testing User Story:
Completing Purchase");
        // Code for testing purchase functionality

        System.out.println("\nAgile Model: Sprints
completed successfully!");

        scanner.close();
    }
}
```

In this example, the Agile Model is represented through three sprints, each focusing on a specific user story. The program outputs messages simulating the tasks performed in each sprint, such as implementing and testing user stories. Note that in a real-world scenario, Agile development involves close collaboration, continuous integration, and frequent releases of working software.

3. **Big Bang Model:**

- No specific process or planning.
- Development begins without a defined process or formal structure.

The Big Bang Model is a software development model characterized by minimal planning and minimal formal processes. Development begins with the identification of requirements, followed by the implementation phase. This model is often used for small projects or proof-of-concept developments. Here's a simple Java program to illustrate the concept of the Big Bang Model:

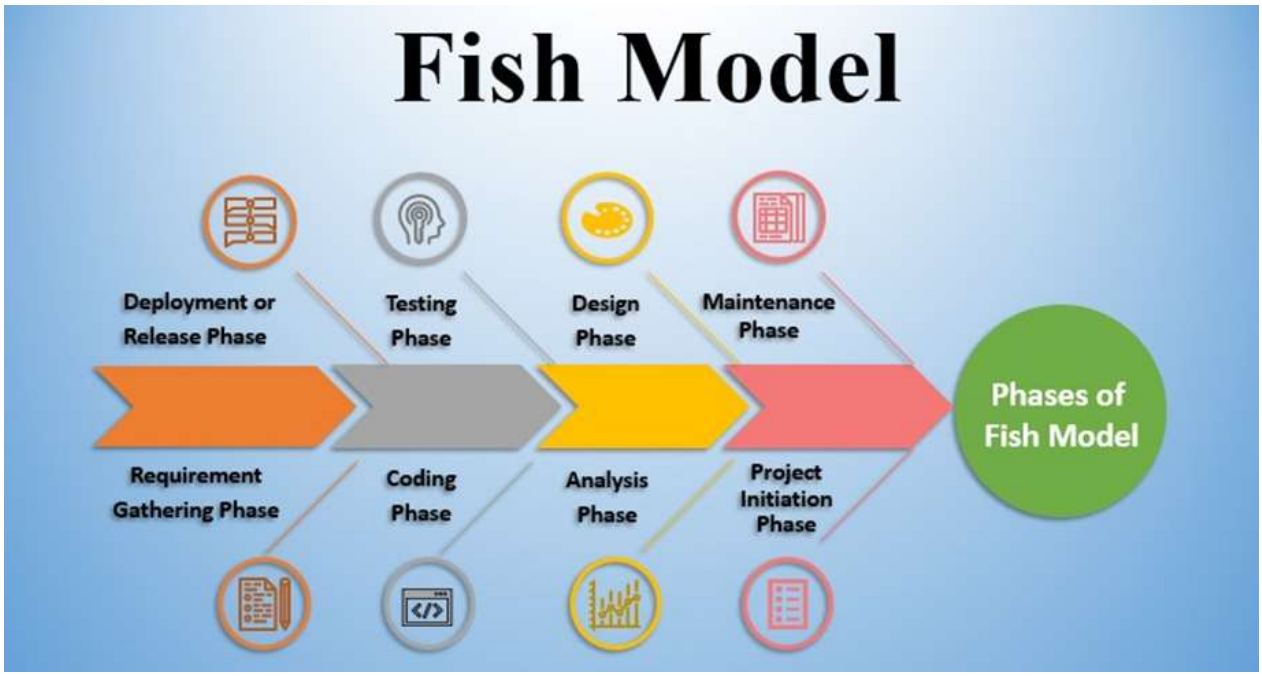
```
import java.util.Scanner;

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

        // Big Bang Model: Requirements
Identification
```

```
        System.out.println("Big Bang Model:
Requirements Identification");
        System.out.println("Identify and gather basic
requirements.");

        // Big Bang Model: Implementation
        System.out.println("\nBig Bang Model:
Implementation");
        System.out.println("Start implementing the
software based on identified requirements.");

        // Simulate continuous development and
improvements
        for (int i = 1; i <= 3; i++) {
            System.out.println("\nIteration " + i +
": Continuous Development");
            System.out.println("Add new features,
make improvements, and adapt to changing
requirements.");
        }

        System.out.println("\nBig Bang Model:
Software developed successfully!");

        scanner.close();
    }
}
```

In this example, the program starts with the identification of basic requirements, followed by the implementation phase. The loop simulates the continuous development and improvement characteristic of the Big Bang Model. Note that in a real-world scenario, this model might lack formalized phases, making it more suitable for small projects or experimental development where requirements may evolve over time.

4. Fish Model:

- Combines elements of the Waterfall and the Prototyping models.
- Iterative development with prototypes and final implementation.



The Fish Model, also known as the Fish Development Model, is a combination of the Waterfall Model and Prototyping Model. It involves a series of phases that are executed in a linear manner, similar to the Waterfall Model, but it also includes prototyping to allow for user feedback and refinement. Here's a simple Java program to represent the Fish Model concept:

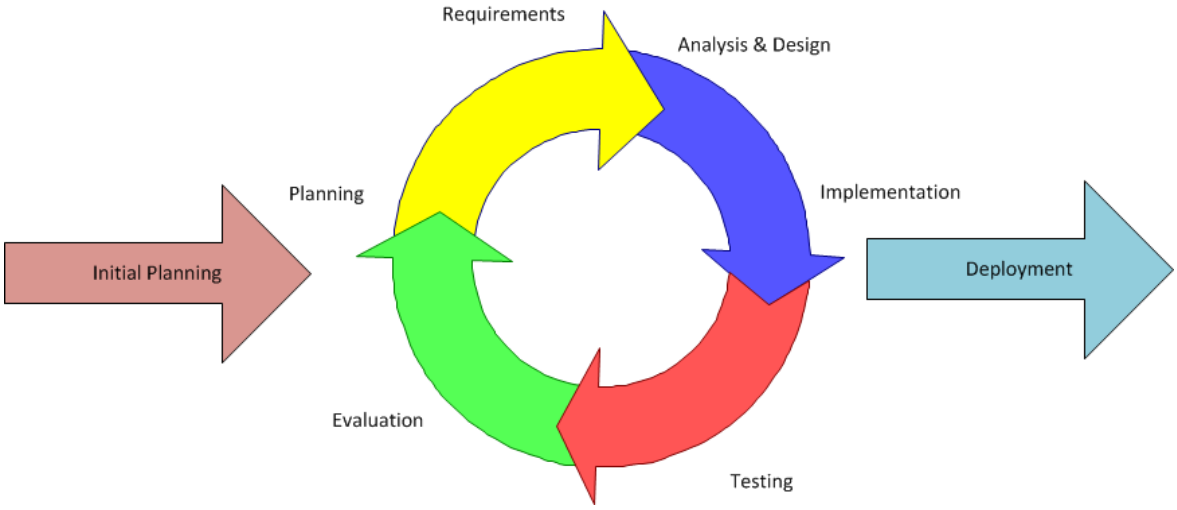
```java
import java.util.Scanner;

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

        // Fish Model: Requirements phase
        System.out.println("Fish Model: Requirements Phase");
        System.out.println("Gather and document initial system
requirements.");

        // Fish Model: Design phase
        System.out.println("\nFish Model: Design Phase");
        System.out.println("Create a basic design based on the
initial requirements.");

        // Fish Model: Prototype phase
        System.out.println("\nFish Model: Prototyping Phase");
        System.out.println("Develop a prototype based on the
basic design.");

        // Gather feedback on the prototype
        System.out.println("Gather feedback from users on the
prototype.");
        System.out.println("Make necessary adjustments based
on user feedback.");

        // Iterate the prototype phase based on feedback
        System.out.println("\nFish Model: Iterative
Prototyping Phase");
        System.out.println("Continue to iterate the
prototyping phase based on user feedback.");

        // Fish Model: Final Implementation phase
        System.out.println("\nFish Model: Final Implementation
Phase");
        System.out.println("Implement the final version based
on the refined prototype.");

        // Fish Model: Testing phase
        System.out.println("\nFish Model: Testing Phase");
        System.out.println("Perform testing to ensure the
final version meets requirements.");

        // Fish Model: Deployment phase
        System.out.println("\nFish Model: Deployment Phase");
        System.out.println("Deploy the software for use.");

        // Fish Model: Maintenance phase
        System.out.println("\nFish Model: Maintenance Phase");
```

```
        System.out.println("Provide ongoing support and make
necessary updates.");

        System.out.println("\nFish Model completed
successfully!");

        scanner.close();
    }
}
```

In this example, the Fish Model is represented by incorporating prototyping phases with user feedback loops. The program outputs messages simulating the tasks performed in each phase, including the iterative nature of the prototyping phase based on user feedback. Keep in mind that the Fish Model may vary in its implementation, and this is a simplified representation for illustration purposes.

5. Hybrid Model:

- Combines aspects of multiple models to meet specific project requirements.
- Tailored to the needs of the project.



The Hybrid Model is an approach that combines elements of different software development models to address specific

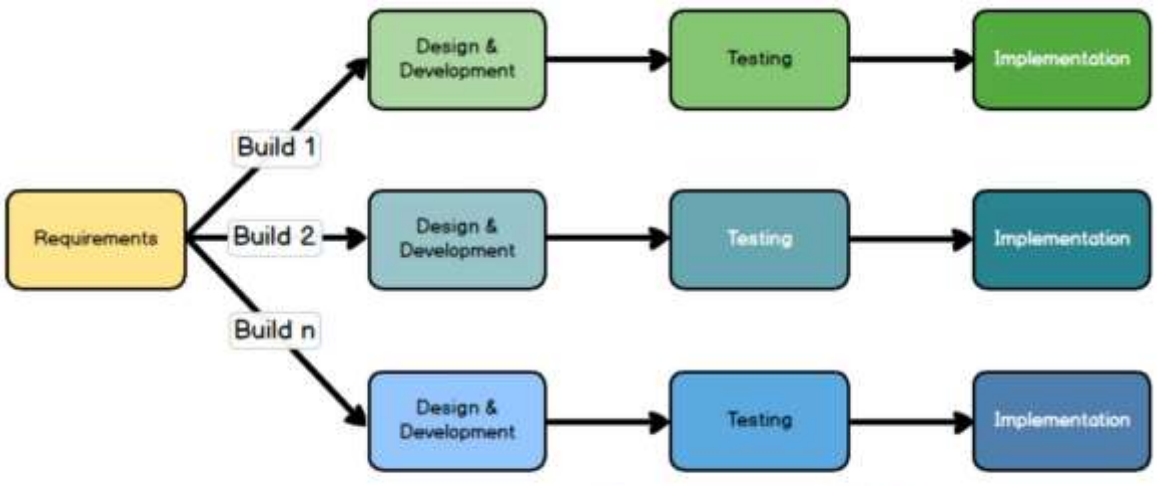
project requirements. It's a flexible and adaptive approach that tailors the development process to the unique characteristics and needs of the project. Here's a simple Java program to illustrate the concept of a Hybrid Model:

```
import java.util.Scanner;

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

        // Hybrid Model: Waterfall-Like Phase
        System.out.println("Hybrid Model: Waterfall-Like
Phase");
        System.out.println("Follow a linear Waterfall-like
process for initial planning and requirements gathering.");

        // Hybrid Model: Iterative Development Phase
        System.out.println("\nHybrid Model: Iterative
Development Phase");
        System.out.println("Adopt an iterative development
process for building and refining the software.");

        // Gather feedback and iterate
        for (int i = 1; i <= 3; i++) {
            System.out.println("\nIteration " + i + ": Gather
feedback and make improvements.");
        }

        // Hybrid Model: Agile-Like Phase
        System.out.println("\nHybrid Model: Agile-Like
Phase");
        System.out.println("Incorporate Agile practices for
continuous collaboration and adaptability.");

        // Conduct sprints with user stories
        for (int sprint = 1; sprint <= 3; sprint++) {
            System.out.println("\nSprint " + sprint + ":
Implement and test user stories.");
        }

        System.out.println("\nHybrid Model completed
successfully!");

        scanner.close();
    }
}
```

In this example, the Hybrid Model is represented by incorporating phases that resemble a Waterfall-like approach for initial planning, followed by an iterative development phase and an Agile-like phase for continuous collaboration. The program outputs messages simulating the tasks performed in each phase. Note that the Hybrid Model is highly customizable, and its implementation may vary based on the specific needs of the project.

6. Incremental Model:

- Divides the project into small, manageable parts or increments.
- Each increment represents a portion of the system's functionality.



Incremental Model

The Incremental Model is an iterative approach to software development where the project is divided into small, manageable parts or increments. Each increment represents a portion of the system's functionality, and new features or functionalities are added in a systematic manner. Here's a simple Java program to illustrate the concept of the Incremental Model:

```
import java.util.Scanner;

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

        // Incremental Model: Increment 1
        System.out.println("Incremental Model: Increment 1");
        System.out.println("Implement and test functionality
for Increment 1.");

        // Gather feedback and make improvements
        System.out.println("Gather feedback and make necessary
improvements based on Increment 1.");

        // Incremental Model: Increment 2
        System.out.println("\nIncremental Model: Increment
2");
        System.out.println("Build upon Increment 1 and
implement functionality for Increment 2.");

        // Gather feedback and make improvements
        System.out.println("Gather feedback and make necessary
improvements based on Increment 2.");

        // Incremental Model: Increment 3
        System.out.println("\nIncremental Model: Increment
3");
        System.out.println("Continue building upon previous
increments and implement functionality for Increment 3.");

        // Gather feedback and make improvements
        System.out.println("Gather feedback and make necessary
improvements based on Increment 3.");
```

```
        System.out.println("\nIncremental Model completed
successfully!");

        scanner.close();
    }
}
```

In this example, the Incremental Model is represented through three increments. The program outputs messages simulating the tasks performed in each increment, including the iterative process of gathering feedback and making improvements based on each increment. Note that in a real-world scenario, each increment might represent a specific set of features or functionalities, and the process of building upon previous increments continues until the complete system is achieved.

7. Iterative Model:

- Involves repeating cycles of the development process.
- Allows for refinement and improvement in each iteration.

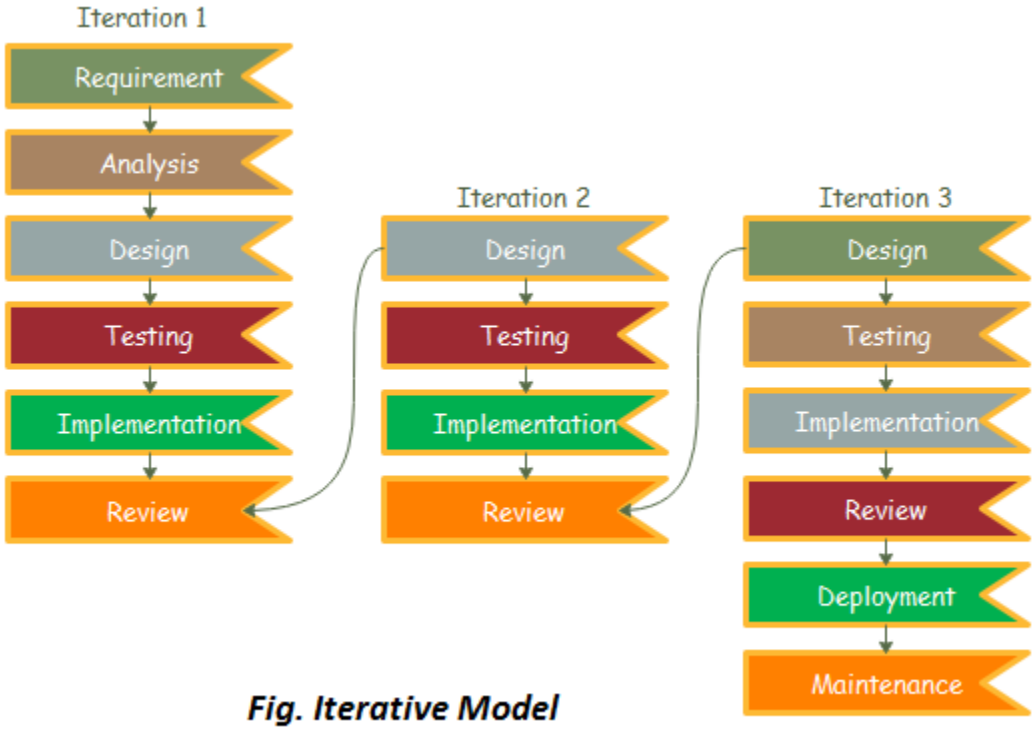


Fig. Iterative Model

The Iterative Model is an approach to software development where the project is broken down into smaller cycles or iterations. Each iteration involves a repetition of the development cycle, allowing for refinement and improvement with each iteration. Here's a simple Java program to illustrate the concept of the Iterative Model:

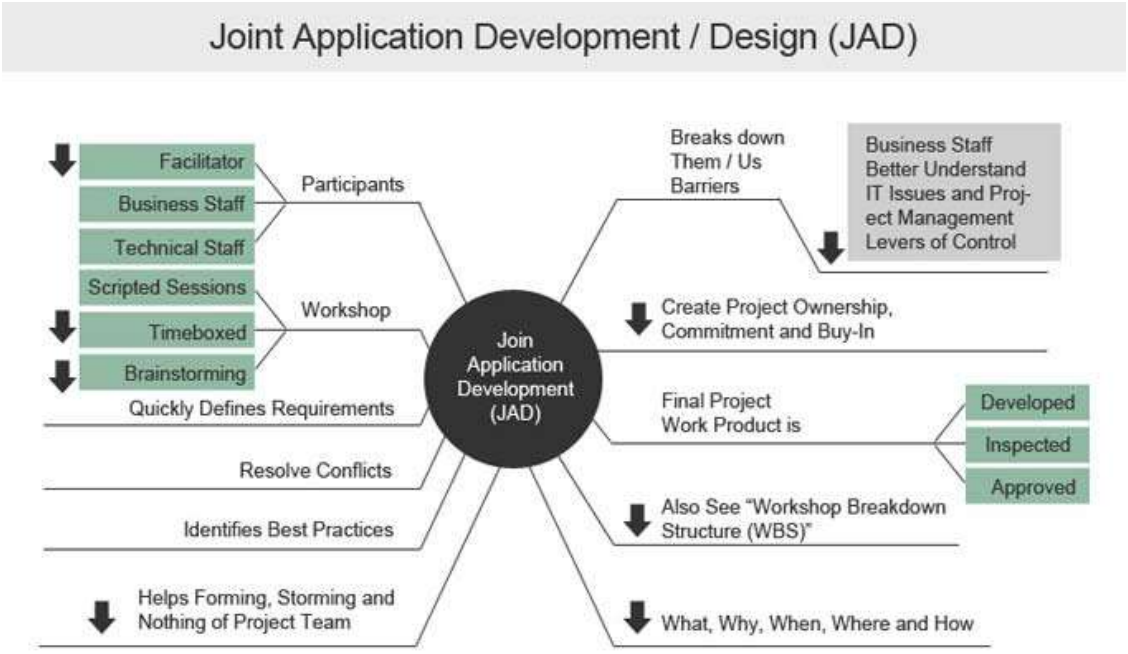
```
import java.util.Scanner;

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

        // Initial requirements gathering
        System.out.println("Iterative Model: Initial
Requirements Gathering");
        System.out.println("Gather and document initial system
requirements.");

        // Iterative development cycle
        for (int iteration = 1; iteration <= 3; iteration++) {
            System.out.println("\nIteration " + iteration + ":
Iterative Development Cycle");

            // Design phase
            System.out.println("Design the system based on
current requirements.");

            // Implementation phase
            System.out.println("Implement the system based on
the designed specifications.");

            // Testing phase
            System.out.println("Test the system to ensure it
meets the requirements.");

            // Gather feedback and refine requirements
            System.out.println("Gather feedback from
stakeholders and refine system requirements.");

            // Continue the next iteration
        }
```

```
        System.out.println("\nIterative Model completed
successfully!");

        scanner.close();
    }
}
```

In this example, the program simulates the iterative development cycle with three iterations. Each iteration includes phases such as requirements gathering, design, implementation, testing, and gathering feedback for refinement. This process is repeated for a specified number of iterations, allowing for continuous improvement and adaptation to changing requirements. In a real-world scenario, the number of iterations and the activities within each iteration would depend on the project's specific needs and goals.

8. JAD Model (Joint Application Development):

- Emphasizes collaborative sessions involving stakeholders, users, and developers.
- Aims to quickly finalize system requirements.



The Joint Application Development (JAD) Model is a collaborative approach to software development that involves workshops and intensive sessions with stakeholders, users, and development teams. The goal is to quickly finalize system requirements and create a shared understanding of the project. Here's a simple Java program to illustrate the concept of the JAD Model:

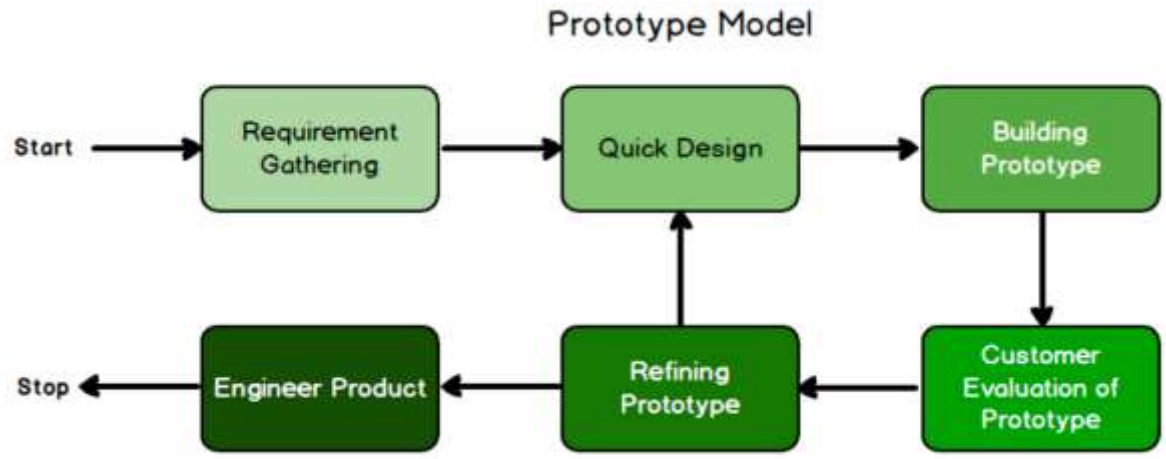
```
import java.util.Scanner;

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

        // JAD Model: Requirements Workshop
        System.out.println("JAD Model: Requirements
Workshop");
        System.out.println("Conduct a collaborative workshop
with stakeholders to gather and finalize system
requirements.");

        // JAD Model: System Design Workshop
        System.out.println("\nJAD Model: System Design
Workshop");
        System.out.println("Collaboratively design the system
based on the finalized requirements.");

        // JAD Model: Prototype Development
        System.out.println("\nJAD Model: Prototype
Development");
        System.out.println("Create a prototype of the system
to visualize and validate the design.");

        // Gather feedback on the prototype
        System.out.println("Gather feedback from stakeholders
on the prototype.");

        // JAD Model: Iterative Refinement
        System.out.println("\nJAD Model: Iterative
Refinement");
        System.out.println("Iteratively refine the
requirements and design based on stakeholder feedback.");
```

```
        // JAD Model: Finalization
        System.out.println("\nJAD Model: Finalization");
        System.out.println("Finalize the system requirements
and design collaboratively.");

        // JAD Model: Implementation and Testing
        System.out.println("\nJAD Model: Implementation and
Testing");
        System.out.println("Implement and test the system
based on the finalized requirements and design.");

        System.out.println("\nJAD Model completed
successfully!");

        scanner.close();
    }
}
```

In this example, the program simulates the JAD Model process, including requirements and design workshops, prototype development, gathering feedback, and iterative refinement. The JAD Model emphasizes collaboration and active involvement of stakeholders throughout the development process to ensure that the final product meets their expectations. In practice, JAD sessions may involve various tools and techniques to facilitate communication and understanding among all involved parties.

9. Prototype Model:

- Involves the creation of an initial, working model of the software.
- Helps in understanding and refining requirements.



The Prototype Model is an approach to software development where a prototype (a preliminary version of the system) is built to demonstrate the system's functionality and gather feedback from users. This iterative process allows for refinement and improvement based on user feedback. Here's a simple Java program to illustrate the concept of the Prototype Model:

```
import java.util.Scanner;

// Interface representing the prototype
interface Prototype {
    void display();
}

// Concrete implementation of the prototype
class ConcretePrototype implements Prototype {
    private String feature;

    public ConcretePrototype(String feature) {
        this.feature = feature;
    }

    @Override
    public void display() {
        System.out.println("Prototype Feature: " + feature);
    }
}

public class PrototypeModelExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
```

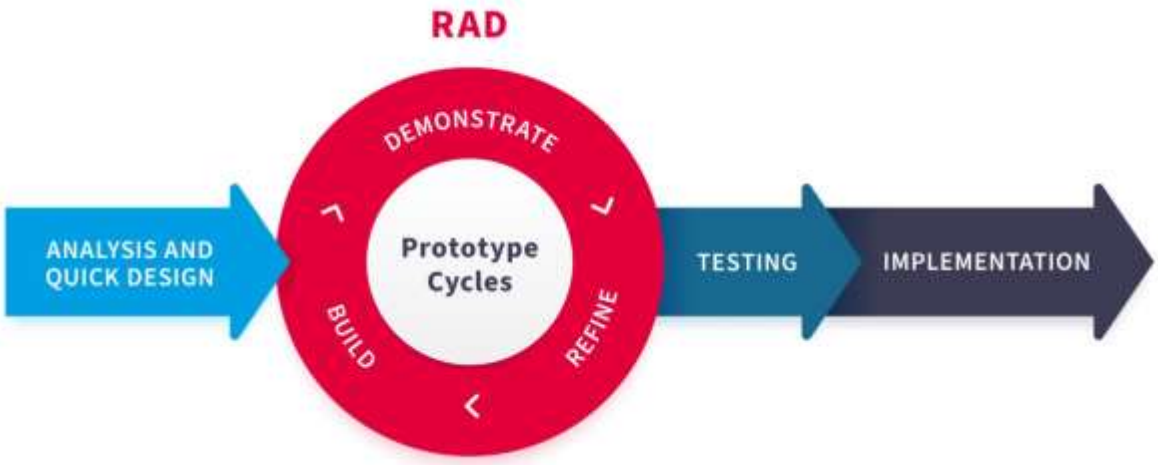
```
        // Prototype Model: Initial Prototype
        System.out.println("Prototype Model: Initial
Prototype");
        System.out.println("Create an initial prototype with
basic features.");

        // Display the initial prototype
        Prototype initialPrototype = new
ConcretePrototype("Basic Functionality");
        initialPrototype.display();

        // Gather feedback and make improvements
        System.out.println("\nGather feedback from users and
stakeholders.");
        System.out.println("Make necessary improvements based
on feedback.");

        // Prototype Model: Iterative Prototyping
        for (int iteration = 1; iteration <= 2; iteration++) {
            System.out.println("\nIteration " + iteration + ":
Iterative Prototyping");

            // Create an improved prototype
            Prototype improvedPrototype = new
ConcretePrototype("Enhanced Functionality");
            improvedPrototype.display();

            // Gather feedback and make further improvements
            System.out.println("Gather feedback from users and
stakeholders.");
            System.out.println("Make necessary improvements
based on feedback.");
        }

        System.out.println("\nPrototype Model completed
successfully!");

        scanner.close();
    }
}
```

In this example, the program simulates the Prototype Model process with an initial prototype and two iterations of iterative prototyping. The Prototype interface defines the common behavior for prototypes, and the ConcretePrototype class represents a specific implementation with a feature. This simple example demonstrates the iterative nature of the Prototype

Model, where feedback is gathered, and improvements are made in successive iterations. In a real-world scenario, the prototype may evolve through multiple iterations based on user feedback and project requirements.

10. The Rapid Application Development (RAD)

- Focuses on rapid prototyping and quick feedback.
- Short development cycles with frequent user feedback.



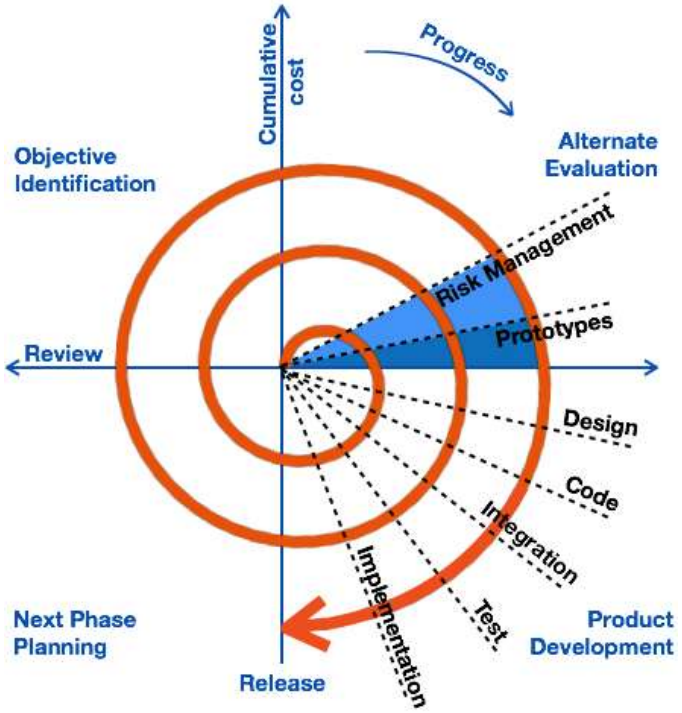
The Rapid Application Development (RAD) Model is an incremental software development process that prioritizes quick development and iteration. It involves close collaboration between developers and end-users, and it often incorporates prototyping to speed up the development cycle. Here's asimple Java program to illustrate the concept of the RAD Model:

```java
import java.util.Scanner;

// Interface representing the RAD prototype
interface RADPrototype {
    void display();
}

// Concrete implementation of the RAD prototype
class ConcreteRADPrototype implements RADPrototype {
    private String feature;

    public ConcreteRADPrototype(String feature) {
        this.feature = feature;
    }

    @Override
    public void display() {
        System.out.println("RAD Prototype Feature: " +
feature);
    }
}

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

        // RAD Model: Iterative Prototyping
        for (int iteration = 1; iteration <= 3; iteration++) {
            System.out.println("RAD Model: Iterative
Prototyping - Iteration " + iteration);

            // Rapidly develop and demonstrate the prototype
            RADPrototype radPrototype = new
ConcreteRADPrototype("Feature Set " + iteration);
            radPrototype.display();

            // Gather feedback and make improvements
            System.out.println("Gather feedback from users and
stakeholders.");
            System.out.println("Make necessary improvements
based on feedback.");
        }

        System.out.println("\nRAD Model completed
successfully!");

        scanner.close();
    }
}
```

In this example, the program simulates the RAD Model process with three iterations of iterative prototyping. The RADPrototype interface defines the common behavior for RAD prototypes, and the ConcreteRADPrototype class represents a specific implementation with a feature. The program demonstrates the rapid development and demonstration of prototypes in successive iterations, gathering feedback, and making improvements based on that feedback. In practice, RAD often involves tight collaboration and communication between developers and end-users to ensure the quick delivery of a functional and satisfactory product.

11. Spiral Model:

- Combines the idea of iterative development with the systematic aspects of the Waterfall model.
- Progresses through multiple iterations known as spirals.



The Spiral Model is a risk-driven software development process model that combines elements of iterative development with aspects of the Waterfall Model. It involves a series of repeated cycles, called spirals, each representing a phase in the software development life cycle. The model is characterized by its emphasis on risk assessment and mitigation throughout the development process. Here's a simple Java program to illustrate the concept of the Spiral Model:

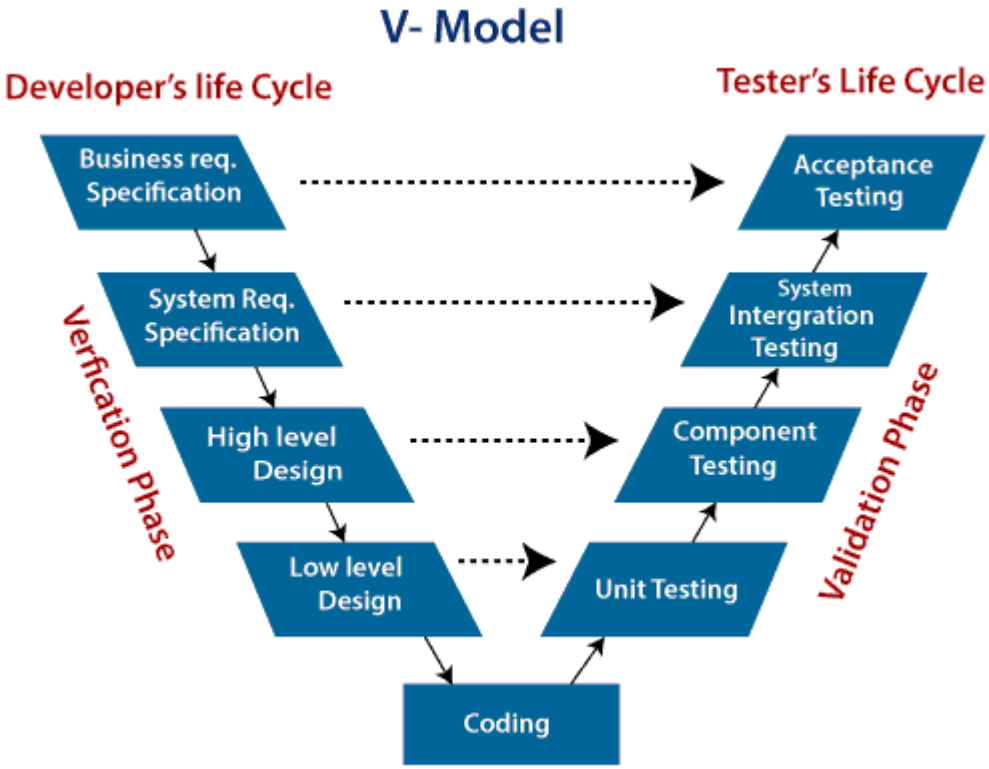
```
import java.util.Scanner;

// Interface representing the spiral phase
interface SpiralPhase {
    void execute();
}

// Concrete implementation of the spiral phase
class RequirementGatheringPhase implements SpiralPhase {
    @Override
    public void execute() {
        System.out.println("Executing Requirement Gathering
Phase");
        // Code for gathering and documenting system
requirements
    }
}

class DesignPhase implements SpiralPhase {
    @Override
    public void execute() {
        System.out.println("Executing Design Phase");
        // Code for creating a detailed design based on
requirements
    }
}

class ImplementationPhase implements SpiralPhase {
    @Override
    public void execute() {
        System.out.println("Executing Implementation Phase");
        // Code for implementing the software based on the
design
    }
}
```

```
class TestingPhase implements SpiralPhase {
    @Override
    public void execute() {
        System.out.println("Executing Testing Phase");
        // Code for testing the software to ensure it meets
requirements
    }
}

// Concrete implementation of the Spiral Model
class SpiralModel {
    private SpiralPhase currentPhase;

    public void executeSpiral() {
        System.out.println("Executing Spiral Model");

        // Execute each phase in a spiral
        executePhase(new RequirementGatheringPhase());
        executePhase(new DesignPhase());
        executePhase(new ImplementationPhase());
        executePhase(new TestingPhase());

        // Additional code for risk assessment and mitigation
can be added here
    }

    private void executePhase(SpiralPhase phase) {
        currentPhase = phase;
        currentPhase.execute();
    }
}

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

        // Create and execute the Spiral Model
        SpiralModel spiralModel = new SpiralModel();
        spiralModel.executeSpiral();

        System.out.println("\nSpiral Model completed
successfully!");

        scanner.close();
    }
}
```

In this example, the program simulates the Spiral Model by defining four phases (Requirement Gathering, Design, Implementation, and Testing) as concrete implementations of the SpiralPhase interface. The SpiralModel class orchestrates the execution of these phases in a spiral, demonstrating the iterative and risk-driven nature of the Spiral Model. In practice, each spiral may involve additional activities such as risk analysis, prototyping, and customer evaluation. The model allows for flexibility and adjustment based on changing requirements and identified risks.

12. V Model:

- Corresponds each stage of development with a testing phase.
- Emphasizes the verification and validation of each phase before moving on.



The V Model, also known as the Verification and Validation Model, is an extension of the traditional Waterfall Model. It emphasizes the relationship between each phase of the development life cycle and its corresponding testing phase. The V Model incorporates testing at each stage, making it a systematic and structured approach to software development. Here's a simple Java program to illustrate the concept of the V Model:

```
import java.util.Scanner;

// Interface representing a development phase
interface DevelopmentPhase {
    void execute();
}

// Concrete implementation of the development phases
class RequirementsPhase implements DevelopmentPhase {
    @Override
    public void execute() {
        System.out.println("Executing Requirements Phase");
        // Code for gathering and documenting system
requirements
    }
}

class DesignPhase implements DevelopmentPhase {
    @Override
    public void execute() {
        System.out.println("Executing Design Phase");
        // Code for creating a detailed design based on
requirements
    }
}

class ImplementationPhase implements DevelopmentPhase {
    @Override
    public void execute() {
        System.out.println("Executing Implementation Phase");
        // Code for implementing the software based on the
design
    }
}
```

```
class TestingPhase implements DevelopmentPhase {
    @Override
    public void execute() {
        System.out.println("Executing Testing Phase");
        // Code for testing the software to ensure it meets
requirements
    }
}

// Concrete implementation of the V Model
class VModel {
    private DevelopmentPhase currentPhase;

    public void executeVModel() {
        System.out.println("Executing V Model");

        // Execute each development phase
        executePhase(new RequirementsPhase());
        executePhase(new DesignPhase());
        executePhase(new ImplementationPhase());

        // Execute each testing phase in reverse order
        executePhase(new TestingPhase());

        // Additional code for validation can be added here
    }

    private void executePhase(DevelopmentPhase phase) {
        currentPhase = phase;
        currentPhase.execute();
    }
}

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

        // Create and execute the V Model
        VModel vModel = new VModel();
        vModel.executeVModel();

        System.out.println("\nV Model completed
successfully!");

        scanner.close();
    }
}
```

In this example, the program simulates the V Model by defining four development phases (Requirements, Design, Implementation, and Testing) as concrete implementations of the DevelopmentPhase interface. The VModel class orchestrates the execution of these phases and their corresponding testing phases in a structured manner. In practice, the V Model helps ensure that testing activities are aligned with the corresponding development activities, promoting early detection and correction of defects.

These models offer different approaches to software development, and the choice of a specific model depends on the project requirements, timeline, and other factors. Many organizations may also adopt a hybrid approach, combining elements from multiple models to suit their needs.