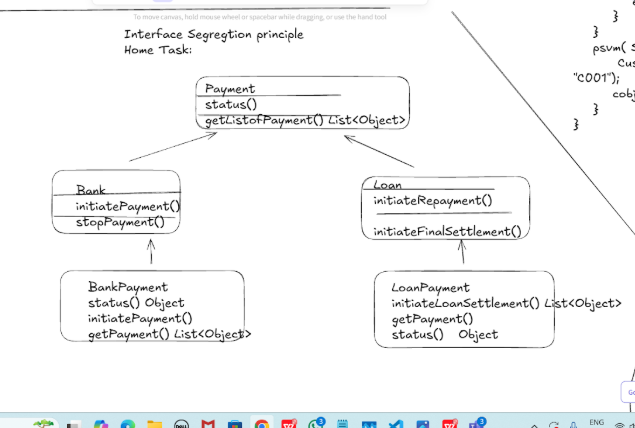
Task1:



package July26;  
  
import java.util.List;  
import java.util.ArrayList;  
  
// Interfaces  
interface IPaymentStatus {  
 Object status();  
}  
  
interface IPaymentHistory {  
 List<Object> getListOfPayment();  
}  
  
interface IBankPayment {  
 void initiatePayment();  
 void stopPayment();  
}  
  
interface ILoanPayment {  
 void initiateRepayment();  
 void initiateFinalSettlement();  
}  
  
// Concrete Class 1: BankPayment  
class BankPayment implements IPaymentStatus, IPaymentHistory, IBankPayment {  
 @Override  
 public Object status() {  
 System.*out*.println("BankPayment Status: SUCCESS");  
 return "SUCCESS";  
 }  
  
 @Override  
 public List<Object> getListOfPayment() {  
 System.*out*.println("Fetching list of bank payments");  
 return new ArrayList<>();  
 }  
  
 @Override  
 public void initiatePayment() {  
 System.*out*.println("Bank payment initiated");  
 }  
  
 @Override  
 public void stopPayment() {  
 System.*out*.println("Bank payment stopped");  
 }  
}  
  
// Concrete Class 2: LoanPayment  
class LoanPayment implements IPaymentStatus, IPaymentHistory, ILoanPayment {  
 @Override  
 public Object status() {  
 System.*out*.println("LoanPayment Status: PENDING");  
 return "PENDING";  
 }  
  
 @Override  
 public List<Object> getListOfPayment() {  
 System.*out*.println("Fetching list of loan repayments");  
 return new ArrayList<>();  
 }  
  
 @Override  
 public void initiateRepayment() {  
 System.*out*.println("Loan repayment initiated");  
 }  
  
 @Override  
 public void initiateFinalSettlement() {  
 System.*out*.println("Final loan settlement initiated");  
 }  
}  
  
// Driver Class  
public class DriverClass3 {  
 public static void main(String[] args) {  
 BankPayment bank = new BankPayment();  
 bank.initiatePayment();  
 bank.status();  
 bank.getListOfPayment();  
 bank.stopPayment();  
  
 System.*out*.println("---");  
  
 LoanPayment loan = new LoanPayment();  
 loan.initiateRepayment();  
 loan.initiateFinalSettlement();  
 loan.status();  
 loan.getListOfPayment();  
 }  
}

output:

Bank payment initiated

BankPayment Status: SUCCESS

Fetching list of bank payments

Bank payment stopped

---

Loan repayment initiated

Final loan settlement initiated

LoanPayment Status: PENDING

Fetching list of loan repayments

Process finished with exit code 0

Task 02:

class Animal {

void sound() {

sout(" sounds of different animals");

}

}

class Cat extends Animal{

@Override

void sound() {

sout(" Meow is the sound of cat");

}

}

class Main{

psvm(String[] args) {

Animal obj = new Cat();

obj.sound(); //Meow is the sound of cat

}

}

issue with Substitution  and Generics

Java Generics -- it has introduced  a challenge - substitution principle...

  is cat a subtype of Animal,  List<cat> is not a subtype of List<Animal>

List<Cat> Cobj = new ArrayList<>();

List<Animal> Aobj = Cobj;  ===// this will give you a wildcard ,

wildcards:

1. unbounded Wildcard

?    ===> any data type if you want to use you can use ?

2. Bounded Wildcards with an upper bound  (? extends Type)

     ===> yopu will use it when you need to accept a type and its own subtypes

3. Bounded Wildcards with an lower bound (? super Type)

     ===> you will use when ypou need accept a type and its super types

Code:

package July26;  
  
import java.util.\*;  
  
class Animal {  
 void sound() {  
 System.*out*.println("Animal sound");  
 }  
}  
  
class Cat extends Animal {  
 void sound() {  
 System.*out*.println("Meow is the sound of cat");  
 }  
}  
  
public class Animals {  
 public static void main(String[] args) {  
 List<Cat> cats = new ArrayList<>();  
 cats.add(new Cat());  
  
 *printAnimalSounds*(cats); // ✅ Using ? extends Animal  
  
 List<Animal> animals = new ArrayList<>();  
 *addCats*(animals); // ✅ Using ? super Cat  
 }  
  
 // Upper bounded: can read Animal or Cat  
 public static void printAnimalSounds(List<? extends Animal> list) {  
 for (Animal animal : list) {  
 animal.sound();  
 }  
 }  
  
 // Lower bounded: can add Cat to Animal or Object  
 public static void addCats(List<? super Cat> list) {  
 list.add(new Cat());  
 }  
}

output:

Meow is the sound of cat

Process finished with exit code 0

Unbounded wildcards:

Task 03:

they are useful when the code does not depends on the actual type parmeter

void printList(List<?>  list) {

for(Object element: list) {

sout (element);

}

}

List<Cat> clist = new ArrayList<>();

clist.add(new Cat());

printList(clist); //

code:

package July26;  
  
import java.util.\*;  
  
// Base class  
class Animal {  
 @Override  
 public String toString() {  
 return "Generic Animal";  
 }  
}  
  
// Subclass  
class Cat extends Animal {  
 @Override  
 public String toString() {  
 return "Cat";  
 }  
}  
  
public class Main {  
  
 // Task 03: Unbounded wildcard method  
 public static void printList(List<?> list) {  
 for (Object element : list) {  
 System.*out*.println(element);  
 }  
 }  
  
 public static void main(String[] args) {  
 List<Cat> clist = new ArrayList<>();  
 clist.add(new Cat());  
 *printList*(clist); // ✅ Accepts List<Cat>, List<Animal>, List<String>, etc.  
  
 List<String> strList = Arrays.*asList*("Hello", "World");  
 *printList*(strList); // ✅ Also works  
 }  
}

output:

Cat

Hello

World

Process finished with exit code 0

Task 04:

Upper Bounded Wildcards

void animalSound(List<? extends Animal> animalList) {

for(Animal elements : animalList

elements.sound();

}

}

List<Cat> cats = new ArrayList<>();

cats.add(new Cat());

animalSound(cats); //meow

code:

package July26;  
  
import java.util.\*;  
  
// Superclass  
class Animal {  
 void sound() {  
 System.*out*.println("Some animal sound");  
 }  
}  
  
// Subclass  
class Cat extends Animal {  
 @Override  
 void sound() {  
 System.*out*.println("Meow");  
 }  
}  
  
public class Mainn {  
  
 // Task 04: Upper bounded wildcard  
 public static void animalSound(List<? extends Animal> animalList) {  
 for (Animal element : animalList) {  
 element.sound();  
 }  
 }  
  
 public static void main(String[] args) {  
 List<Cat> cats = new ArrayList<>();  
 cats.add(new Cat());  
  
 *animalSound*(cats); // ✅ Works due to upper bound  
 }  
}

output:

Meow