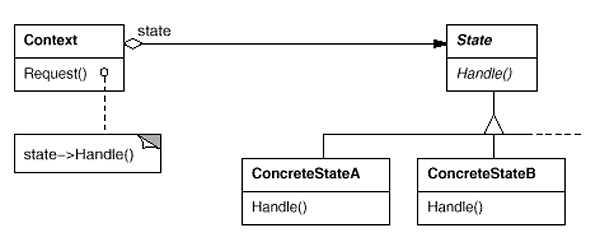
**State Design Pattern -Behavioural- 2022**

GOF : **Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.**



public interface PackageState {  
 public void updateState(DeliveryContext ctx);  
}

public class DeliveryContext {  
 private PackageState currentState;  
  
 public DeliveryContext(PackageState currentState) {  
 this.currentState = currentState;  
  
 if (currentState == null) {  
 this.currentState = new Acknowledged();  
 }  
 }  
  
 public PackageState getCurrentState() {  
 return currentState;  
 }  
  
 public void setCurrentState(PackageState currentState) {  
 this.currentState = currentState;  
 }  
 public void update() {  
 currentState.updateState(this);  
 }  
}

public class Acknowledged implements PackageState {  
  
 *//Business logic and state transition* @Override  
 public void updateState(DeliveryContext ctx) {  
 System.*out*.println("Package is acknowledged !!");  
 ctx.setCurrentState(new Shipped());  
 }  
}

public class Shipped implements PackageState {  
  
 *//Business logic and state transition* @Override  
 public void updateState(DeliveryContext ctx) {  
 System.*out*.println("Package is shipped !!");  
 ctx.setCurrentState(new InTransition());  
 }  
}

public class InTransition implements PackageState {  
  
 *//Business logic and state transition* @Override  
 public void updateState(DeliveryContext ctx) {  
 System.*out*.println("Package is in transition !!");  
 ctx.setCurrentState(new Delivered());  
 }  
}

public class Delivered implements PackageState {  
  
 *//Business logic* @Override  
 public void updateState(DeliveryContext ctx) {  
 System.*out*.println("Package is delivered!!");  
 }  
}

public class Main {  
 public static void main(String[] args) {  
 DeliveryContext ctx = new DeliveryContext(null);  
  
 ctx.update();  
 ctx.update();  
 ctx.update();  
 ctx.update();  
 }  
}