**Chain of Responsibility**

**Definition**

Let more than one object handle a request without their knowing each other. Pass the request to chained objects until it has been handled.

**Where to use & benefits**

* One request should be handled by more than one object.
* Don't know which object should handle a request, probably more than one object will handle it automatically.
* Reduce coupling.
* Flexible in handling a request.
* Related patterns include
  + [Composite](http://www.javacamp.org/designPattern/composite.html), which a chain of responsibility pattern is often applied in conjunction with.

**Example**

The Java Servlet filter framework is an example of chain of resposibility design. Note that the chain.doFilter() is the method that should be called to make the chain roll. If the subclass missed it, the whole chain would be stopped or blocked.

Java exception handling is another example of chain of responsibility design. When an error occurs, the exception call will look for a handling class. If there is no handler, the super Exception class will be called to throw the exception. Otherwise, the handler class will handle it.

Here comes a simple example, just to show how chain of responsibility works. Whenever you spend company's money, you need get approval from your boss, or your boss's boss. Let's say, the leadership chain is:

Manager-->Director-->Vice President-->President

The following is a command line program to check who is responsible to approve your expenditure.

import java.io.\*;

abstract class PurchasePower {

protected final double base = 500;

protected PurchasePower successor;

public void setSuccessor(PurchasePower successor){

this.successor = successor;

}

abstract public void processRequest(PurchaseRequest request);

}

class ManagerPPower extends PurchasePower {

private final double ALLOWABLE = 10 \* base;

public void processRequest(PurchaseRequest request ) {

if( request.getAmount() < ALLOWABLE )

System.out.println("Manager will approve $"+ request.getAmount());

else

if( successor != null)

successor.processRequest(request);

}

}

class DirectorPPower extends PurchasePower {

private final double ALLOWABLE = 20 \* base;

public void processRequest(PurchaseRequest request ) {

if( request.getAmount() < ALLOWABLE )

System.out.println("Director will approve $"+ request.getAmount());

else

if( successor != null)

successor.processRequest(request);

}

}

class VicePresidentPPower extends PurchasePower {

private final double ALLOWABLE = 40 \* base;

public void processRequest(PurchaseRequest request) {

if( request.getAmount() < ALLOWABLE )

System.out.println("Vice President will approve $" + request.getAmount());

else

if( successor != null )

successor.processRequest(request);

}

}

class PresidentPPower extends PurchasePower {

private final double ALLOWABLE = 60 \* base;

public void processRequest(PurchaseRequest request){

if( request.getAmount() < ALLOWABLE )

System.out.println("President will approve $" + request.getAmount());

else

System.out.println( "Your request for $" + request.getAmount() + " needs a board meeting!");

}

}

class PurchaseRequest {

private int number;

private double amount;

private String purpose;

public PurchaseRequest(int number, double amount, String purpose){

this.number = number;

this.amount = amount;

this.purpose = purpose;

}

public double getAmount() {

return amount;

}

public void setAmount(double amt){

amount = amt;

}

public String getPurpose() {

return purpose;

}

public void setPurpose(String reason) {

purpose = reason;

}

public int getNumber(){

return number;

}

public void setNumber(int num) {

number = num;

}

}

class CheckAuthority {

public static void main(String[] args){

ManagerPPower manager = new ManagerPPower();

DirectorPPower director = new DirectorPPower();

VicePresidentPPower vp = new VicePresidentPPower();

PresidentPPower president = new PresidentPPower();

manager.setSuccessor(director);

director.setSuccessor(vp);

vp.setSuccessor(president);

//enter ctrl+c to kill.

try{

while (true) {

System.out.println("Enter the amount to check who should approve your expenditure.");

System.out.print(">");

double d = Double.parseDouble(new BufferedReader(new InputStreamReader(System.in)).readLine());

manager.processRequest(new PurchaseRequest(0, d, "General"));

}

}catch(Exception e){

System.exit(1);

}

}

}

| C:\ Command Prompt  C:\> javac CheckAuthority.java  C:\> java CheckAuthority  Enter the amount to check who should approve your expenditure.  >500  Manager will approve $500.0  Enter the amount to check who should approve your expenditure.  >5000  Director will approve $5000.0  Enter the amount to check who should approve your expenditure.  >11000  Vice President will approve $11000.0  Enter the amount to check who should approve your expenditure.  >30000  Your request for $30000.0 needs a board meeting!  Enter the amount to check who should approve your expenditure.  >20000  President will approve $20000.0  Enter the amount to check who should approve your expenditure.  >  C:\> |
| --- |

You may redo it using interface instead of abstract class.

The [composite](http://www.javacamp.org/designPattern/composite.html) pattern is often used with chain of responsibility. That means a class may contain the related class that may handle the request.