the base URI/server for the API ( such as <http://myapi.com/resources/>)

operation set/request methods, most notably: GET, PUT, POST, or DELETE

Response data, usually json format, xml was popular years ago

For instance, at Expresscopy, we had an api server <https://api.expresscopy.com>/product that provides product services/resources, I implemented a price calculator that retrieved product prices from the resource:

Send: GET /

[https://api.expresscopy.com/product/price/?key=dd6c323162a9e778fa73b56455269a6e&binderyArray=,,,&siteId=2&productId=1&quantity=100&format=js&callback=ec.pricing.\_addEstimate](https://api.expresscopy.com/product/price/?key=dd6c323162a9e778fa73b56455269a6e&binderyArray=,,,&siteId=2&productId=1&quantity=100&format=js&callback=ec.pricing._addEstimate&cb=1372464209021)

Here we have API server: <https://api.expresscopy.com/product/price/>

Using GET method sending the reqyest:

?key=dd6c323162a9e778fa73b56455269a6e&binderyArray=,,,&siteId=2&productId=1&quantity=100&format=js&callback=ec.pricing.addEstimate

GET back

ec.pricing.\_addEstimate({"response":{"minQty":"10","price":"69.00","unitPrice":"0.690","postageUnitPrice":"0.330","bindery":"","totalBinderyPrice":"0.00","nextPriceBreak":"250"}});

Here we get the pricing data Response call back in json format.

Abstract classes allow you to provide default base functionality for the subclasses. To define a class as Abstract, the keyword *abstract* is to be used . Functions in abstract classes must be declared as protected or public to be inherited.

The child class ‘extends’ the base class to implement new functions that’s not in the abstract classes.

Abstract class Tickets {

Private $row, $column, $type;

**public** **function** setSeat($row, $column) {

$this->row = $row;

$this->column = $column;

}

public function ticketType($type){

$this->type = $type;

}

**public** abstract **function** getPrice();

}

We have a child class SymphoniesTickets that inherent the ticket class

**class** SymphoniesTickets **extends** tickets {

**private** $price;

//seating ticket

**public** getTicket($type, $row, $column, $price) {

parent::setType($type);

parent::setSeat($row, $column);

$this->price = $price;

}

**public** **function** getPrice() {

//more logic here

return $this->price;

}

}

Similarly, a general admission AdmissionTickts that inherent the ticket class

**class** admissionTickets **extends** tickets {

**private** $price;

//seating ticket

**public** getTicket($type = ‘admission’, $row, $column, $price) {

parent::setType($type);

parent::setSeat(0,0);

$this->price = $price;

}

**public** **function** getPrice() {

//more logic here

return $this->price;

}

}

3.To explain these concepts into plain English, many resources are available online for instance:

<http://nicolasgallagher.com/css-cascade-specificity-inheritance/> gives a very precise explanation.

<http://iam.colum.edu/phuber/week2/CSScheatSheet.pdf> gives a more in depth explanations.

Understand the concepts are important, hands on trial and error sometimes provids a quicker way to trouble shoot the issues than trying to calculating the speciality.