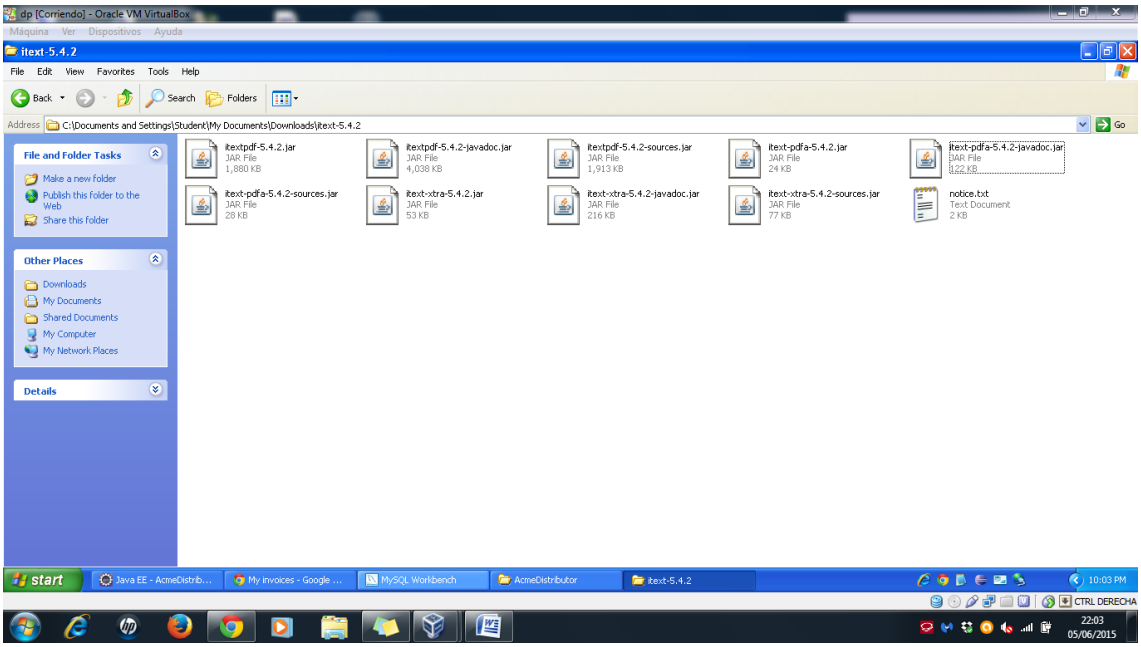
Documento para generar una factura en PDF.

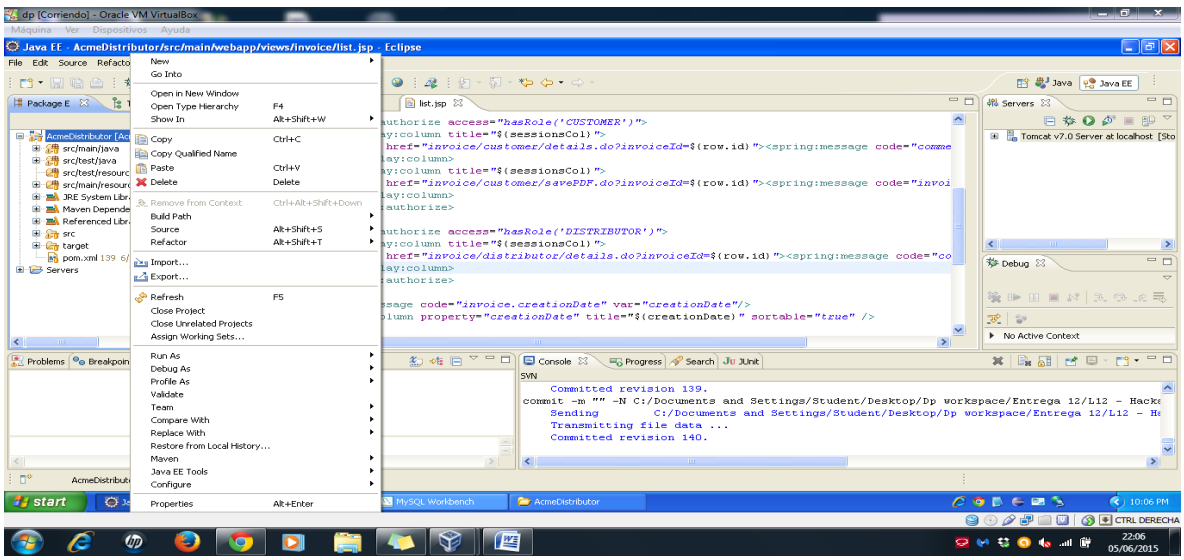
En este documento voy a explicar los pasos previos a realizar, necesarios para que la aplicación genere correctamente un archivo PDF para una factura elegida, siempre que se esté registrado como cliente en la aplicación.

Lo primero es la descarga de la librería que vamos a utilizar para llevar a cabo la creación del archivo en cuestión. En nuestro caso usaremos iText 5.4.2. Para facilitar este paso hemos adjuntado la librería al proyecto, junto con los JARs que necesitamos.

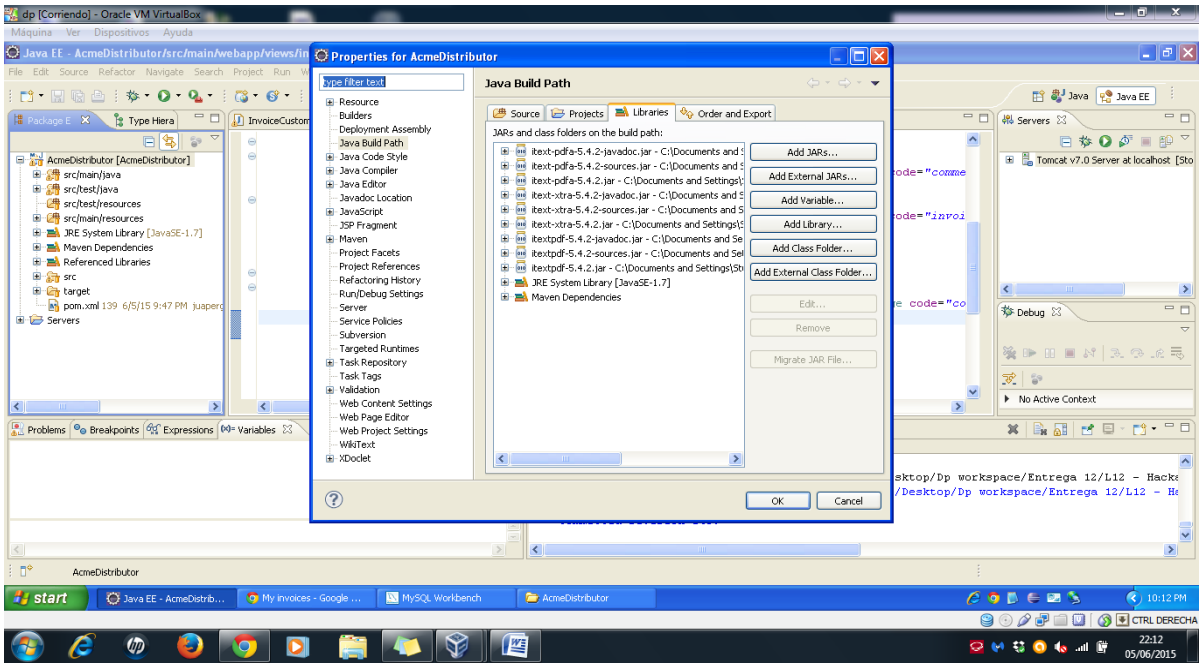


A continuación añadiremos dichos JARs al proyecto, para ello una vez tengamos importado el proyecto en Eclipse, debemos seguir la siguiente ruta:

1.Click derecho sobre el proyecto y pulsar en "Properties".

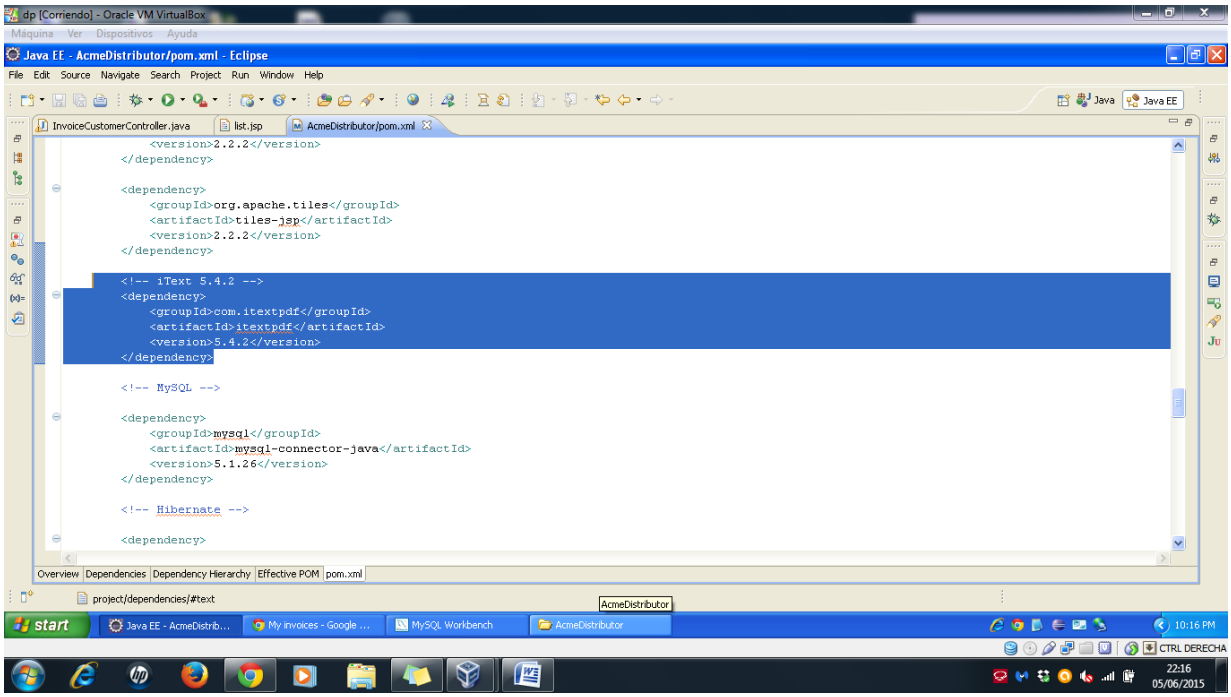


2.Una vez clickeado aparecerá una ventana emergente "Properties for …". Hay que pulsar en "Java Build Path" y a continuación en "Add External JARs…" buscar entonces los JARs en tu PC e importarlos. Finalizar pulsando "OK".

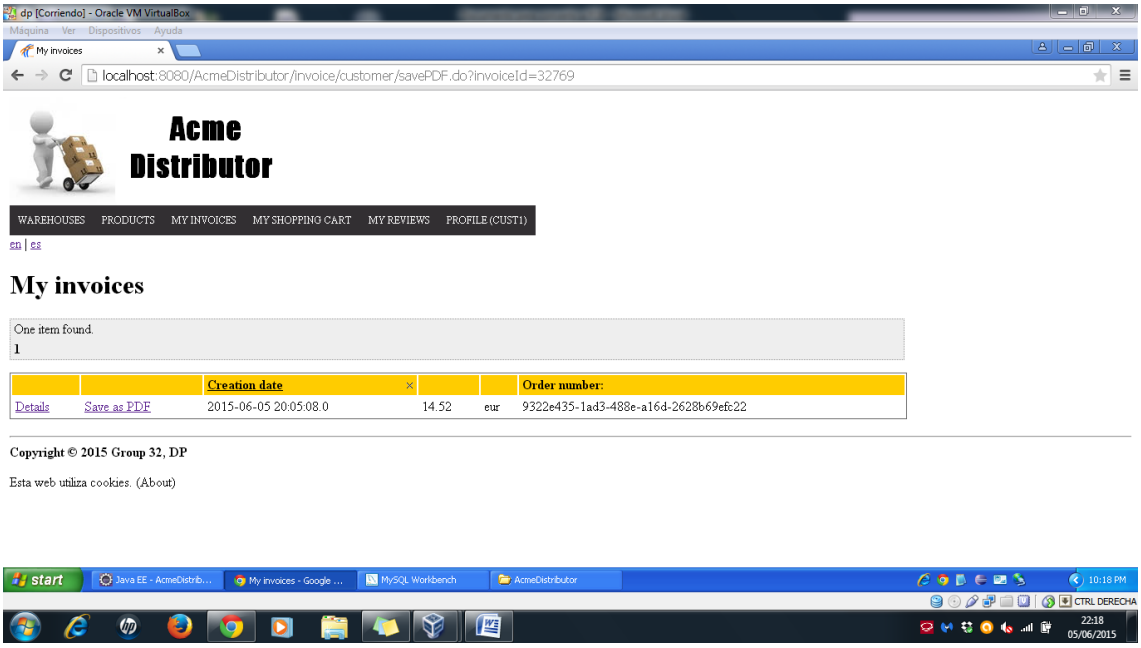


3. Abrir el fichero "pom.xml" y añadir si no lo tiene ya insertado el código :

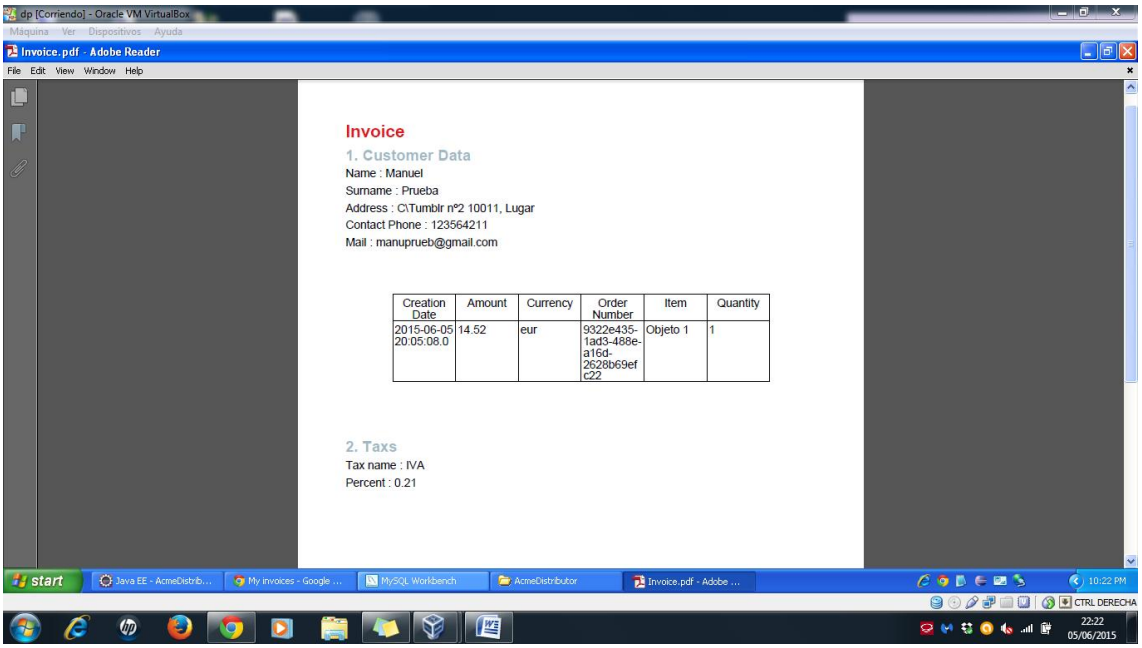
```
<dependency>
        <groupId>com.itextpdf</groupId>
        <artifactId>itextpdf</artifactId>
        <version>5.4.2</version>
</dependency>
```



4. Una vez hecho esto hay que ejecutar la aplicación en el Tomcat y registrarse como Customer y pulsar en el menú en "My invoices" o "Mis facturas" dependiendo del idioma en el que se esté corriendo la aplicación:



5.Pulsar en "Save as PDF" y nos guardará la factura elegida en formato PDF por defecto en la ruta "C:/Documents and Settings/Students/Desktop" con el nombre Invoice.pdf



Bibliografía:

http://hmkcode.com/itext-pdf-java/

http://www.ibm.com/developerworks/library/os-javapdf/#fig5

Google Maps JavaScript API

NOTA: Para la implementación de esta API, ha sido necesario añadir dos atributos a la clase de dominio Warehouse. En concreto, los atributos son:

- **latitude:**Double
- **longitude:**Double

Lo primero que haremos para poder trabajar con los objetos que define la API de Google Maps será importar su librería.

```
<script type="text/javascript"
src="http://maps.googleapis.com/maps/api/js></script>
```

A dicha llamadas podremos añadirle entre otras cosas:

```
key=API_KEY
```

```
https:
```

Una clave que proporciona Google al desarrollador si se desea implementar el uso de Maps de cara al público y de una forma segura, dada la envergadura de nuestra aplicación y el hecho de que será ejecutada en local, no le daremos uso.

```
sensor=true/false
```

Sirve para avisar al navegador de que se usará los sensores del dispositivo para determinar su ubicación, en nuestro caso lo definiremos a true, pues usaremos dicha funcionalidad.

A continuación definiremos una zona para usar funciones y métodos propios de javascript.

```
<script type="text/javascript">
```

Zona que cerraremos una vez acabada la ejecución de dichos métodos.

```
</script>
```

Crearemos un div para albergar el mapa que crearemos posteriormente, div al que aplicaremos estilos vía CSS o directamente con un tag CSS en HTML/JSP.

```
<div id="map_container"></div>

<style type="text/css">

div#map_container{
```

```
            width:70%;
            height:500px;
            margin-left: auto;
            margin-right: auto;
        }
    </style>
```

Ahora pasaremos a crear un objeto literal donde definiremos las propiedades que tendrá nuestro mapa.

```
    var myOptions = {
        zoom: 12,
        center: latlng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    };
```

Como podemos apreciar definimos el nivel de zoom que tendrá el mapa por defecto, centraremos dicho mapa en las coordenadas que nos resulte conveniente y usando la api elegir de un enumerado el tipo de mapa a mostrar.

Una vez tengamos la base, podremos invocar una llamada para crear el mapa, colocándolo en el div que hemos preparado para el y con las opciones que previamente configuramos.

```
    var map = new google.maps.Map(
    document.getElementById("map_container") , myOptions);
```

Ahora solo nos faltara cargar el mapa una vez que la página es cargada.

De haber definido el proceso de creación podemos usar un escuchador de eventos para el DOM, de este modo al ejecutarse la carga de la página este llamara a nuestra función.

```
        google.maps.event.addDomListener(window,'load', loadMap);
```

De lo contrario, podemos usar una alternativa similar, en este caso cuando se cree el body, este ejecutara nuestra función.

```
        <body onload="loadMap()">
        …
        </body>
```

Para definir ventanas de información y marcadores seguiremos pasos muy similares a los usados para la definición del mapa, primero crearemos objetos que contengan las propiedades y a continuación invocaremos las llamadas para colocarlos en el mapa.

```
var warehouseOptions = {

    map: map,

    position: latlng,

    content:

    '<h1>'+"${name}"+'</h1>'+

    '<p><spring:message code= "warehouse.distributor.name"/>:'

    + "${distName}" +'</p>'+

    '<p><spring:message code= "warehouse.contactPhone"/>:'

    + "${phone}" +'</p>'+

    '<p><spring:message code="warehouse.email"/>:'

    + "${email}" +'</p>'+

    '<p><spring:message code="warehouse.address"/>:'

    + "${address}" +'</p>'+

    '<a href="item/listByWarehouse.do?warehouseId=${id}">
    <spring:message code="warehouse.listItems"/></a>'

};
```

Aprovecharemos el controlador para mostrar información dinámicamente, enviando desde este último las variables que queremos mostrar, de modo que nuestro marcador contenga información relevante y contextual.

Capturaremos dichas variables con:

```
"${variable}"
```

Asegurándonos del lado del controlador que la variable que queremos enviar comparte el nombre.

Por otra parte, con la id del Warehouse que estamos mostrando en ese momento aprovecharemos para montar una url apuntando al stock de items de dicho almacen, de modo que se facilite la navegabilidad de cara al usuario.

Con sus opciones creadas, pasamos a invocar el método.

```
var warehouseInfWin = new
google.maps.InfoWindow(warehouseOptions);
```

Aprovecharemos otra de las funcionalidades para atar dicha InfoWindows a un marcador, de modo que si el usuario cierra la InfoWindows pueda volver a abrirla.

Creamos para ello un marcador en la misma ubicación:

```
var marker = new google.maps.Marker({
      position: latlng,
      map: map,
      title: "${name}"
});
```

Y un escuchador asociado a este y al InfoWindows en cuestión que se ocupara de gestionar su apertura:

```
google.maps.event.addListener(marker,'click',function() {
      warehouseInfWin.open(map,marker);
});
```

Finalmente, crearemos un marcador en la posición de nuestro usuario usando la librería para hacer una petición de permiso.

```
if(navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(
            function(position) {
      var pos = new google.maps.LatLng(position.coords.latitude,
      position.coords.longitude);
      var marker = new google.maps.Marker({
            position: pos,
            map: map,
            title: 'My location'
      });
}, function(){
```

```
        handleNoGeolocation(true);
    });
    }else{
        handleNoGeolocation(false);
    }
```

Esto comprobara si el navegador soporta la petición de localización, de ser así solicitará al usuario permiso para obtener la posición.

Dado dicho permiso, crearemos una variable donde almacenaremos latitud y longitud, para posteriormente ubicar un marcador allí.

En caso de que se deniegue la solicitud, se invocara a una función que se ocupara de mostrar un aviso, en nuestro caso hemos dejado la llamada a esta ultima por comodidad, sin embargo el rechazo del permiso por parte del usuario no nos afecta de cara a la creación y uso del mapa, de modo que no hemos considerado la opción de mostrar un aviso en dicho caso.