En este apartado vamos a integrar paso a paso un proyecto de ejemplo con Babel. El proyecto consiste en un conversor de pulgadas a centímetros. El código fuente incluye sintaxis específica de ES2015 y funcionalidades aún no disponibles oficialmente en el lenguaje. Por este motivo necesitamos Babel para que nuestro proyecto sea retrocompatible.

Todo el código del proyecto se encuentra disponible en el repositorio curso-react-babel.

Código fuente

El proyecto se compone de una única clase definida en `index.js` :

```
// Se encarga de la conversión de unidades
class InchConverter {
  // Base de la conversión
  static BASE = 2.54;

  // Toma el valor en centímetros y lo convierte a pulgadas
  static convertCmToInch = value => {
    return value / InchConverter.BASE;
  }
  // Toma el valor en pulgadas y lo convierte a centímetros
  static convertInchToCm = value => {
    return InchConverter.BASE * value;
  }
}

// Exportamos la clase
export default InchConverter;
```

Esta clase hace uso de funcionalidades que aún no están en el lenguaje como las propiedades de clase vistas en el apartado anterior. Si intentamos utilizar la clase desde node v6.9.1 obtenemos un error, ya que `node` no es capaz de interpretar el código fuente.

```
$ node
> var InchConverter = require('./index');
/Users/angelmm/projects/curso-react-babel/index.js:4
  static BASE = 2.54;
              ^
SyntaxError: Unexpected token =
```

Comencemos con la integración de Babel.

Instalación de Babel

Babel se distribuye en un paquete NPM. Lo primero que necesitamos es instalar `node` y `npm`. En la propia web de node podéis descargar distintos paquetes de instalación. Personalmente, os recomiendo instalar nvm. Este es un gestor de versiones de `node`, por lo que podremos tener varias versiones y cambiar entre ellas según los requisitos de nuestros proyectos.

Una vez tenemos instalado `node` y `npm`, creamos una nueva carpeta e inicializamos el proyecto. `npm init` nos crea nuestro fichero `package.json` con la información y las dependencias de nuestro proyecto. Podéis rellenar los campos a vuestro gusto.

```
mkdir curso-react-babel
npm init
```

Ahora vamos a instalar Babel. El paquete `npm` se llama `babel-cli` y nos proporciona su línea de comandos o *CLI*. Con ello ya podremos acceder a `babel` desde el terminal.

```
npm install --save-dev babel-cli
```

El parámetro `--save-dev` guarda el paquete como dependencia de desarrollo o `devDependencies`. Guardamos Babel en esta sección del fichero `package.json` porque solo es útil para compilar el proyecto. Una vez que tengamos el fichero compilado ya no será necesario.

```
-   "homepage": "https://github.com/Angelmmiguel/curso-react-babel#readme"
+   "homepage": "https://github.com/Angelmmiguel/curso-react-babel#readme",
+   "devDependencies": {
+     "babel-cli": "^6.18.0"
+   }
  }
```

Hay que tener en cuenta que `babel` no está instalado de manera global, sino local al proyecto. Por ello, si ejecutamos directamente `babel` en la consola, nos devolverá un error. Babel recomienda este tipo de instalación porque cada proyecto puede utilizar una versión diferente de Babel.

> While you can install Babel CLI globally on your machine, it's much better to install it locally project by project. - BabelJS

Para ejecutar Babel de manera sencilla, incluimos el script para compilar el proyecto en el fichero `package.json`. Este script toma `index.js`, lo compila y guarda el resultado en la carpeta `dist`.

```
  "main": "index.js",
  "scripts": {
```

```
+    "build": "babel index.js -d dist",
     "test": "echo \"Error: no test specified\" && exit 1"
   },
```

Una vez tenemos instalada la dependencia y configurado el fichero `package.json` ya podemos compilar nuestro proyecto mediante el comando `npm run build`. Otra manera de ejecutar Babel es mediante su fichero binario ubicado en `./node_modules/.bin/babel`. Mi recomendación es que siempre definái los `scripts` en el fichero `package.json`. De esta manera siempre tendréis los distintos comandos de vuestro proyecto en el mismo sitio, por lo que será más fácil de utilizar y no olvidaréis ningún parámetro.

Probemos a compilar el proyecto.

```
$ npm run build

> curso-react-babel@1.0.0 build
> babel index.js -d dist

SyntaxError: index.js: Unexpected token (4:14)
  2 | class InchConverter {
  3 |   // Base de la conversion
> 4 |   static BASE = 2.54;
    |              ^
  5 |
  6 |   // Toma el valor en centimetros y lo convierte a pulgadas
  7 |   static convertCmToInch = value => {
```

Obtenemos el mismo error que al ejecutar el proyecto con `node`. Esto es debido a que Babel aún no tiene las suficientes “herramientas” para compilar nuestro código. Necesitamos agregar distintos *presets* y *plugins* para poder compilar nuestro proyecto.

Presets y plugins

Los plugins son librerías que compilan nuevos operadores o sintaxis en código retrocompatible. Esta es la base de Babel, pues cada propuesta que requiere compilar estas nuevas características requiere de un plugin que modifique el código.

Los presets son conjuntos de plugins definidos por Babel. Estos se corresponden con versiones del lenguaje como ES2015 y ES2016. También existe un preset de React, que necesitaremos más adelante.

En la web de Babel tenéis todos plugins y presets disponibles. Para este proyecto vamos a incluir dos presets y un plugin:

- Preset latest: Incluye todas las funcionalidades disponibles en ES2015, ES2016 y ES2017. Es el preset recomendado por babel.
- Preset stage-3: Funcionalidades que están en el último paso antes de ser agregadas al lenguaje.
- Plugin transform-class-properties: Esta funcionalidad aún está en `stage-2`, pero la necesitamos para poder definir propiedades estáticas.

El primer paso para utilizar estos plugins y presets es instalarlos en nuestro proyecto. Ambos tipos se distribuyen como paquetes `npm`.

```
npm install --save-dev babel-preset-latest babel-preset-stage-2 babel-plugin-transform-class-properties
```

Una vez instalados, tenemos que configurar Babel para que los utilice en la compilación de nuestro proyecto. La forma más sencilla es crear el fichero `.babelrc` con nuestra configuración.

```
{
  "presets": [
    "latest",
    "react",
    "stage-3"
  ],
  "plugins": [
    "transform-class-properties"
  ]
}
```

Cabe destacar que el orden de los presets y plugins afecta al proceso de compilación. Cada uno de estos elementos lee el código fuente y devuelve una nueva versión que es entregada al siguiente. Por ejemplo, en el caso de los presets, los `stage-X` deben de ejecutarse antes de que los presets de versiones de ECMAScript como `latest` o `es2016`.

Si os fijáis en el fichero de configuración, el preset `latest` está situado antes que `stage-3`. Esto es debido a que **el orden de lectura de los presets es al revés**. Es decir, `stage-3` será el primer preset. No obstante, **en el caso del array de plugins el orden si corresponde con su posición en el array**. En la sección de "Ordering" de Babel tenéis más información al respecto.

Tras agregar los distintos plugins y presets y configurar Babel, ya podemos compilar nuestro proyecto:

```
$ npm run build

> curso-react-babel@1.0.0 build
```

```
> babel index.js -d dist

index.js -> dist/index.js
```

Una vez compilada la librería ya podemos probarla en `node` :

```
$ node
> var InchConverter = require('./dist/index').default;
> InchConverter.convertCmToInch(10)
3.937007874015748
```