



Ep1 : Blue J

crée par **Yongkang Zou et Yassine Kafraoui**



Dans une ville animée, il y a un concessionnaire automobile appelé **Java Auto**, spécialisé dans la vente de voitures de haute qualité et de systèmes audio innovants pour voitures. Le propriétaire de ce concessionnaire est un programmeur passionné et un amateur de voitures, nommé Jeff.

Un jour, Jeff décide d'incorporer sa compréhension approfondie des classes Java dans le processus de vente de voitures. Il crée une classe Java appelée "**Voiture**", comprenant divers attributs et méthodes tels que "**PrixVoiture**", "**MarqueVoiture**", etc. Pour ajouter des fonctionnalités uniques au système audio de la voiture, il crée également une classe appelée "**AudioSystem**", comprenant des attributs tels que "**PrixAudio**", "**MarqueAudio**", etc.

Grâce à l'utilisation ingénieuse des classes Java, Java Auto devient l'un des concessionnaires automobiles les plus populaires de la ville. Les clients louent la conception remarquable des classes Java, rendant le processus d'achat de voitures plus simple et plus amusant. Chaque fois que quelqu'un achète une voiture, Jeff présente personnellement les concepts des classes Java, montrant aux clients comment les voitures et les systèmes audio sont soigneusement conçus dans le paradigme de la programmation orientée objet en Java.

Dans l'histoire de Java Auto, les classes Java ne sont pas simplement une partie du code, mais plutôt une arme secrète dans le processus de vente de voitures. Ce concessionnaire automobile est renommé pour son concept unique de classes Java, offrant aux clients une expérience d'achat de voiture sans précédent.

Quelques années plus tard, à mesure que le temps passe, Java Auto continue de prospérer, et le fils de Jeff, Jack, grandit également. Peu à peu, il développe un intérêt pour la concession automobile de son père et aspire à devenir programmeur. En tant que père bienveillant et programmeur qualifié, Jeff s'apprête à enseigner personnellement à Jack comment il a créé ce concessionnaire automobile au style geek et avant-gardiste, main dans la main, dévoilant les coulisses de la construction de cet établissement unique.



I. Première itération - BlueJ

Jeff est passionné par l'importance de la conception des classes Java. Il comprend que la conception d'une classe solide est la base de l'ensemble du système. C'est pourquoi il opte pour **BlueJ** comme environnement de développement intégré (IDE) pour Jack, afin de l'aider à clarifier les relations entre les classes "**Voiture**" et "**AudioSystem**".

BlueJ lui permet de visualiser de manière claire et structurée les interactions entre ces deux classes clés. Jeff utilise cette plateforme pour créer des diagrammes de classes, facilitant ainsi la compréhension de Jack quant à la manière dont les différentes classes interagissent et se complètent mutuellement dans le système global.

Étape 1 : Jeff a ouvert son vieux dossier de favoris sur le web, longtemps négligé, et a retrouvé l'adresse de téléchargement de **BlueJ** pour Jack : <http://www.bluej.org/>

Download and Install

Version 5.2.1, released 10 October 2023 (Fix installer issues, [see more](#))

Windows



Requires 64-bit Windows,
Windows 7 or later. Also
available: [Standalone zip](#)
suitable for USB drives.

Mac OS X



Requires OS X 10.12 or later.

Ubuntu/Debian



Requires 64-bit, Debian
buster or Ubuntu 18.10 or
later. There is also a [flatpak](#)
available (maintained by
someone else).

Other



Please read the [Installation
instructions](#). (Works on most
platforms with Java/JavaFX
17 support).

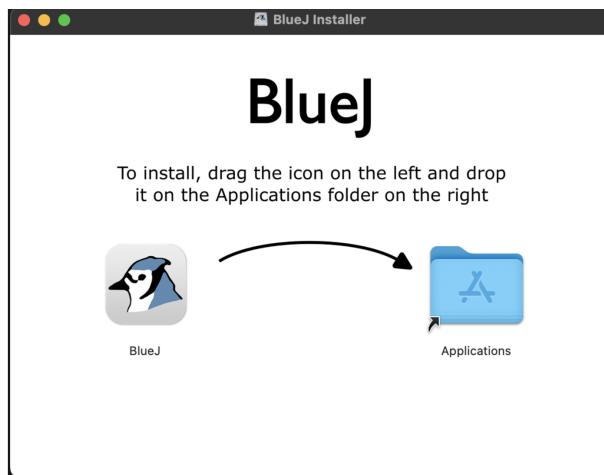
Note: BlueJ now uses Java 11+, which requires a 64-bit operating system, which 95+% of users will have.
For 32-bit operating systems, [download BlueJ 4.1.4](#) instead.

Download [previous versions](#) or [old source code archives](#).

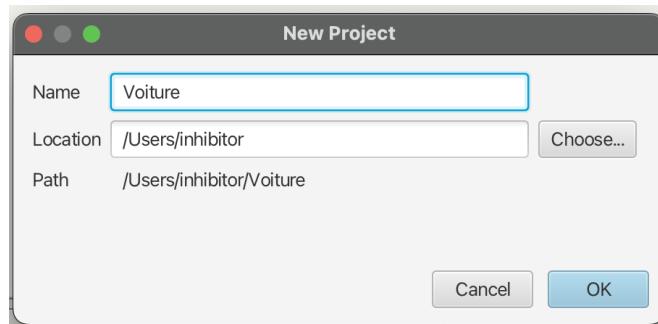
The source code for BlueJ is now available on [Github](#).

The copyright for BlueJ is held by M. Kölling and J. Rosenberg.
BlueJ is available under the GNU General Public License version 2 with the
Classpath Exception ([full license text](#), [licenses for third party libraries](#)).

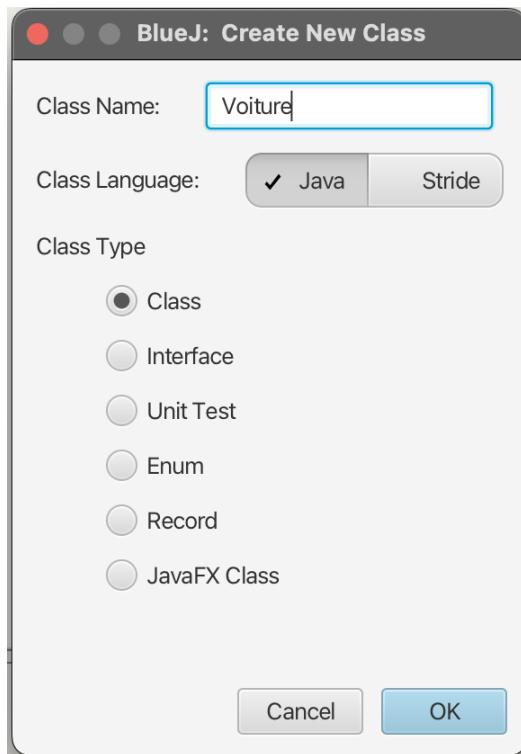
Étape 2 : Sous la direction de son père, Jack a ouvert le téléchargement pour le système d'exploitation **Mac OS** :



Étape 3 : Une fois le téléchargement terminé, Jack n'a pas pu attendre et a immédiatement créé un nouveau projet qu'il a nommé "**Voiture**". Sous la tutelle de son père, Jack s'est plongé dans l'univers de la programmation avec enthousiasme, prêt à explorer les concepts de classes Java et à donner vie à son projet.

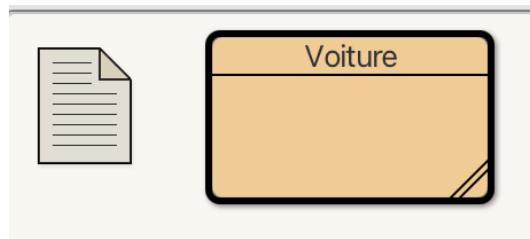


Étape 4 : Rapidement, avant même que son père puisse donner la prochaine étape, Jack crée déjà sa première classe "**Voiture**".

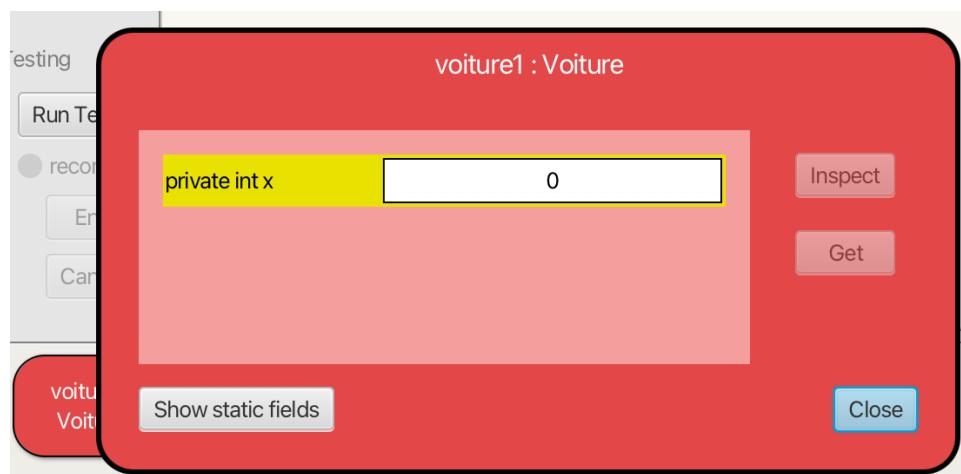


Étape 5 : "Pourquoi y a-t-il des barres obliques dans cette boîte ?" demande Jack, perplexe, en regardant son père. "C'est parce que cette classe est encore vide",

explique Jeff. Il clique sur "Compile". Maintenant, il est prêt à enseigner officiellement à Jack comment utiliser BlueJ pour écrire sa première classe Java.



Étape 6 : Jeff tente de faire observer à Jack ce qu'un objet peut contenir lorsqu'on en crée un. "Pourquoi il n'y a rien à l'intérieur ?" demande Jack.



Étape 7 : "Parce qu'il est nécessaire de définir les propriétés de la classe par nous-mêmes", explique Jeff. Il montre ensuite comment ajouter deux attributs, "***prixVoiture***" et "***marqueVoiture***", en les configurant par défaut avec la voiture la plus vendue, la "***Tesla***", et son prix correspondant de **10000** euros. En même temps, il met en place leurs constructeurs ainsi qu'une méthode simple de calcul du prix réduit des voitures, appelée "***promoVoiture***".

```

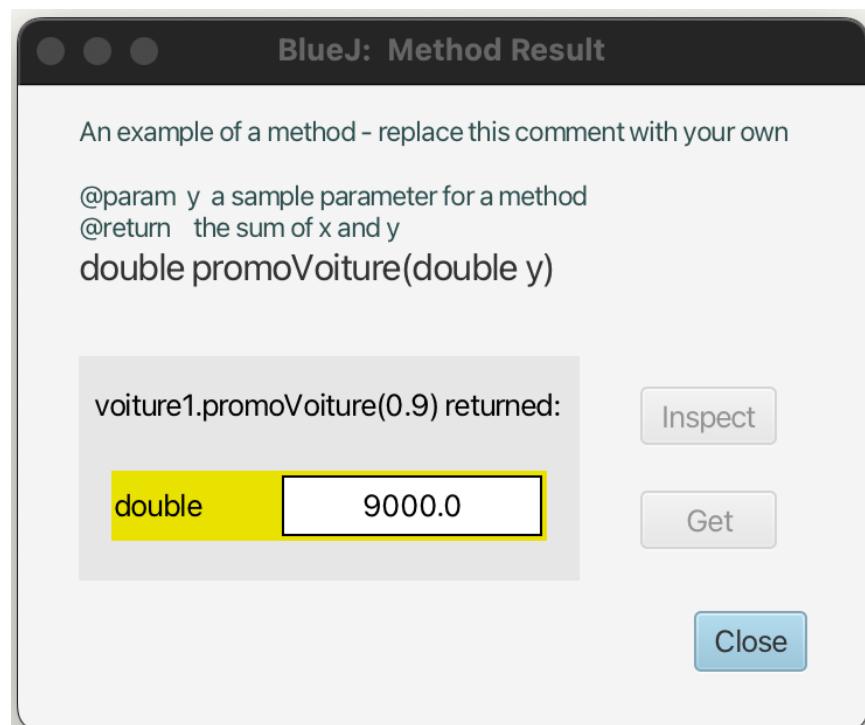
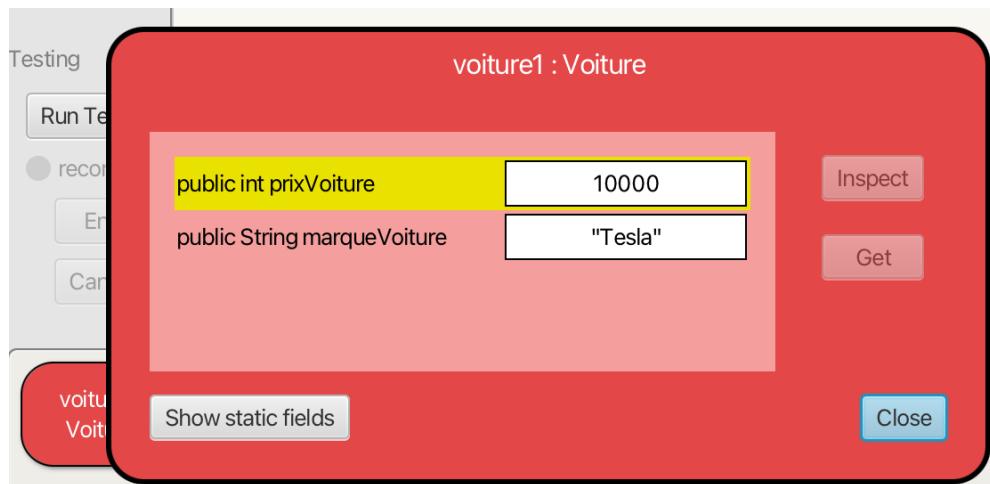
public class Voiture
{
    // instance variables
    private double prixVoiture;
    private String marqueVoiture;

    /**
     * Constructor for objects of class voiture
     */
    public Voiture()
    {
        // initialise instance variables
        prixVoiture = 10000;
        marqueVoiture = "Tesla";
    }
    public double getPrixVoiture()
    {
        return prixVoiture;
    }
    public void setPrixVoiture(double prixVoiture)
    {
        this.prixVoiture = prixVoiture;
    }
    public String getMarqueVoiture()
    {
        return marqueVoiture;
    }
    public void setMarqueVoiture(String marqueVoiture)
    {
        this.marqueVoiture = marqueVoiture;
    }

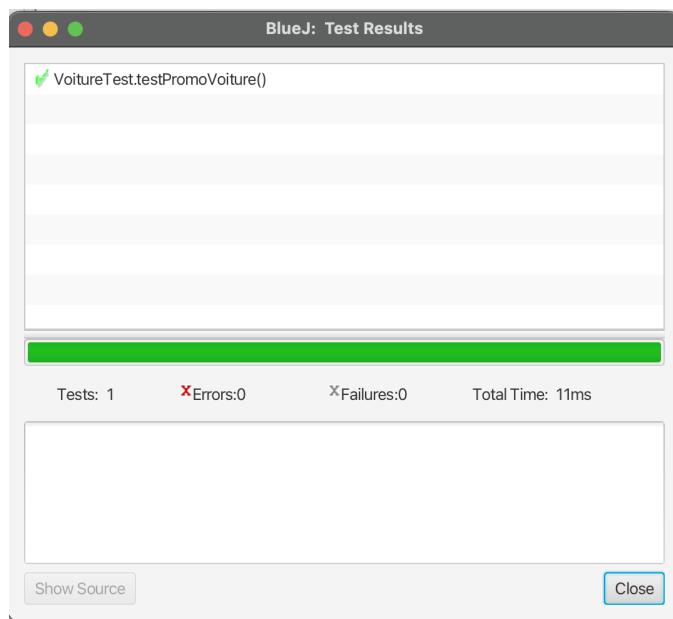
    public double promoVoiture(double y)
    {
        // put your code here
        return prixVoiture*y;
    }
}

```

Étape 8 : À ce moment-là, Jeff demande à Jack de créer un autre objet pour voir et de tester la nouvelle méthode "***promoVoiture***" qu'ils viennent de créer.



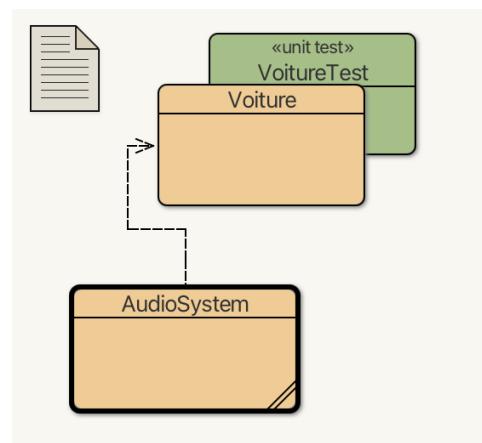
Étape 9 : "Vite, créons un **test unitaire** pour essayer ça. Établir une méthode et la tester immédiatement devient une bonne habitude pour les programmeurs." "D'accord, papa."



```
@Test
public void testPromoVoiture()
{
    Voiture voiture1 = new Voiture();
    assertEquals(9000, voiture1.promoVoiture(0.9), 0.1);
}
```

Étape 10 : "Maintenant, créons la deuxième classe, '**AudioSystem**'", dit Jeff. "Dans notre concession, nous vendons souvent des systèmes audio avec les voitures. Bien que parfois, des clients ne veuillent pas installer un système audio dans leur nouvelle voiture, ou souhaitent personnaliser un système audio pour une voiture qu'ils ont achetée ailleurs. C'est pourquoi, lors de la création de la classe '**AudioSystem**', j'ai inclus la classe '**Voiture**' que nous venons de créer en tant que paramètre de base."

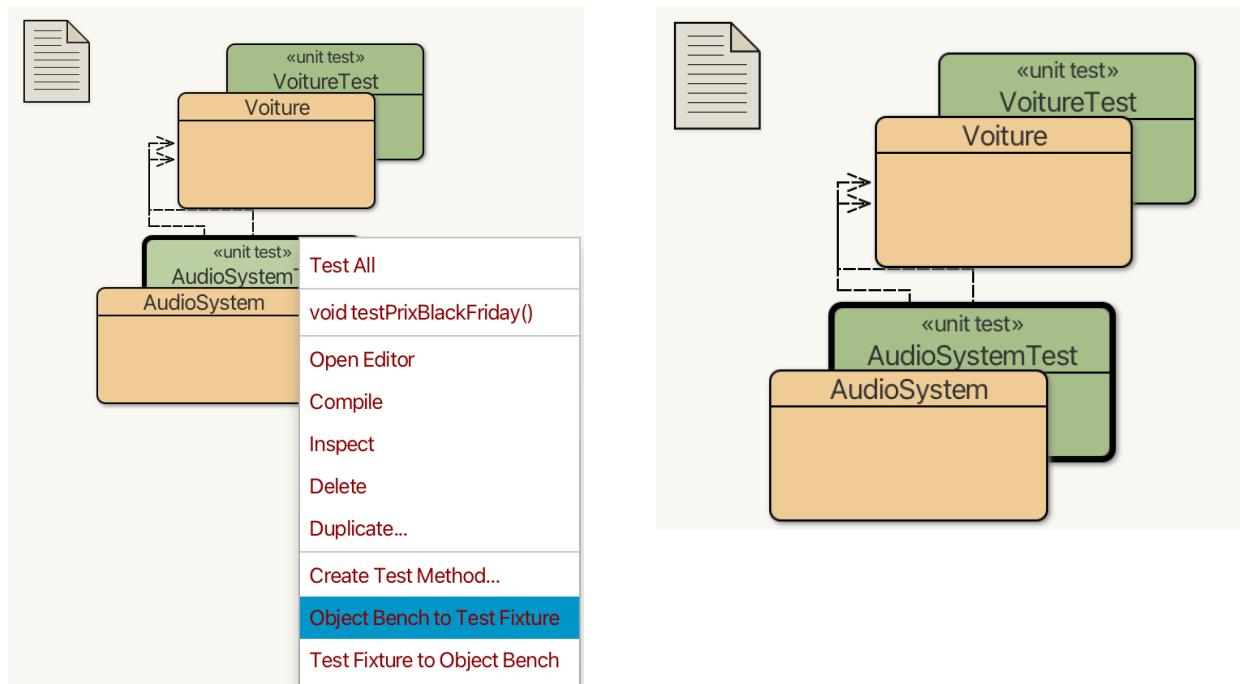
```
public class AudioSystem
{
    // instance variables
    private double prixAudio;
    private String marqueAudio;
    private Voiture voiture;
```



Étape 11 : "Tu dois savoir que, en plus d'un modèle de vente novateur, offrir aux clients les meilleures remises possibles est le secret de notre flux continu de clients dans notre magasin." Tout en parlant, Jeff crée une méthode appelée "***PrixBlackFriday***", qui offre des réductions groupées pour les systèmes audio embarqués et les voitures pendant la période du Vendredi noir.

```
public double PrixBlackFriday(Voiture voiture, double promo)
{
    return prixAudio*promo + voiture.promoVoiture(promo);
}
```

Étape 12 : "Maintenant, je vais t'enseigner comment tester linstanciation simultanée de ces deux classes, les lier et les utiliser comme éléments de mise en place (fixture) dans la configuration de la classe de test."



```

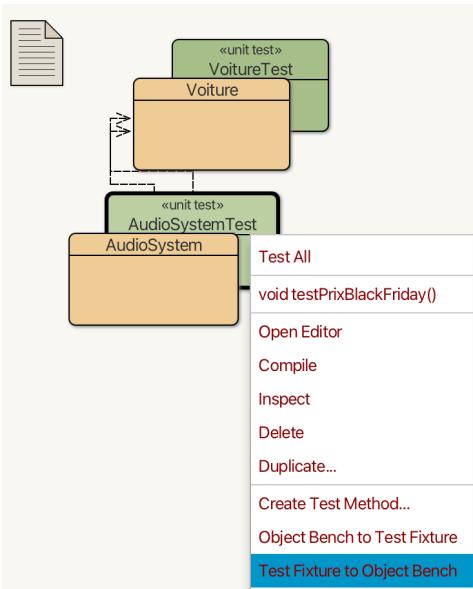
public class AudioSystemTest
{
    private Voiture voiture1;
    private AudioSystem audioSys1;

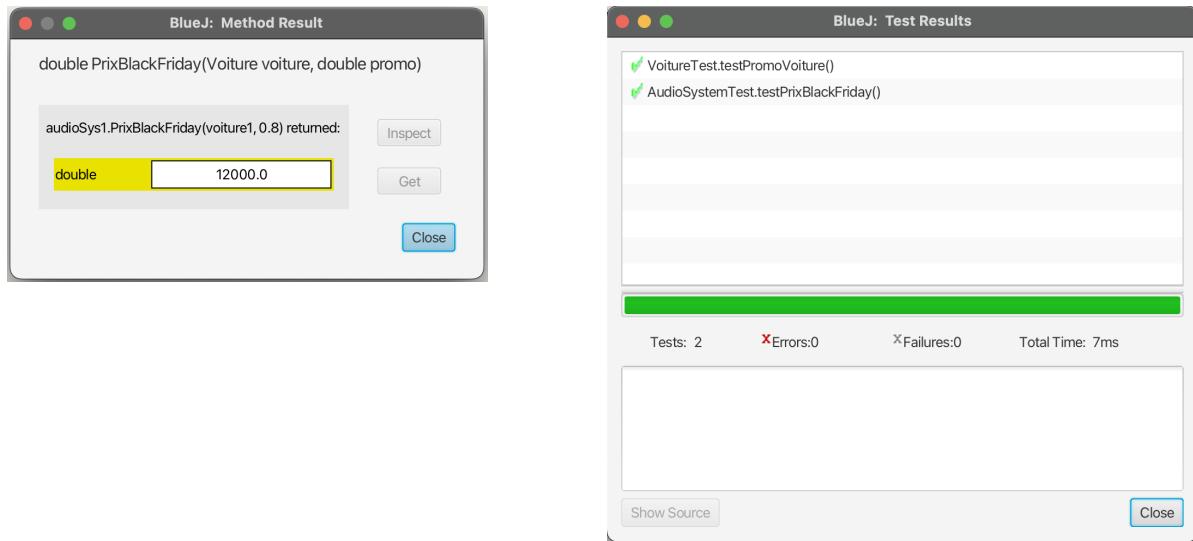
    /**
     * Default constructor for test class AudioSystemTest
     */
    public AudioSystemTest()
    {
    }

    /**
     * Sets up the test fixture.
     *
     * Called before every test case method.
     */
    @BeforeEach
    public void setUp()
    {
        voiture1 = new Voiture();
        audioSys1 = new AudioSystem();
    }
}

```

Étape 13 : "Maintenant, testons la méthode '**PrixBlackFriday**' que nous venons de créer", dit Jeff.





```
@Test
public void testPrixBlackFriday()
{
    assertEquals(12000, audioSys1.PrixBlackFriday(voiture1, 0.8), 0.1);
}
```

"Pas mal du tout," dit Jeff en regardant Jack qui vient de terminer la tâche. "On reprend ça demain pour la suite." Voyant que Jack commence à bâiller, Jeff caresse la tête de Jack avec satisfaction en disant, "Tu as bien travaillé. Allons-y demain."

