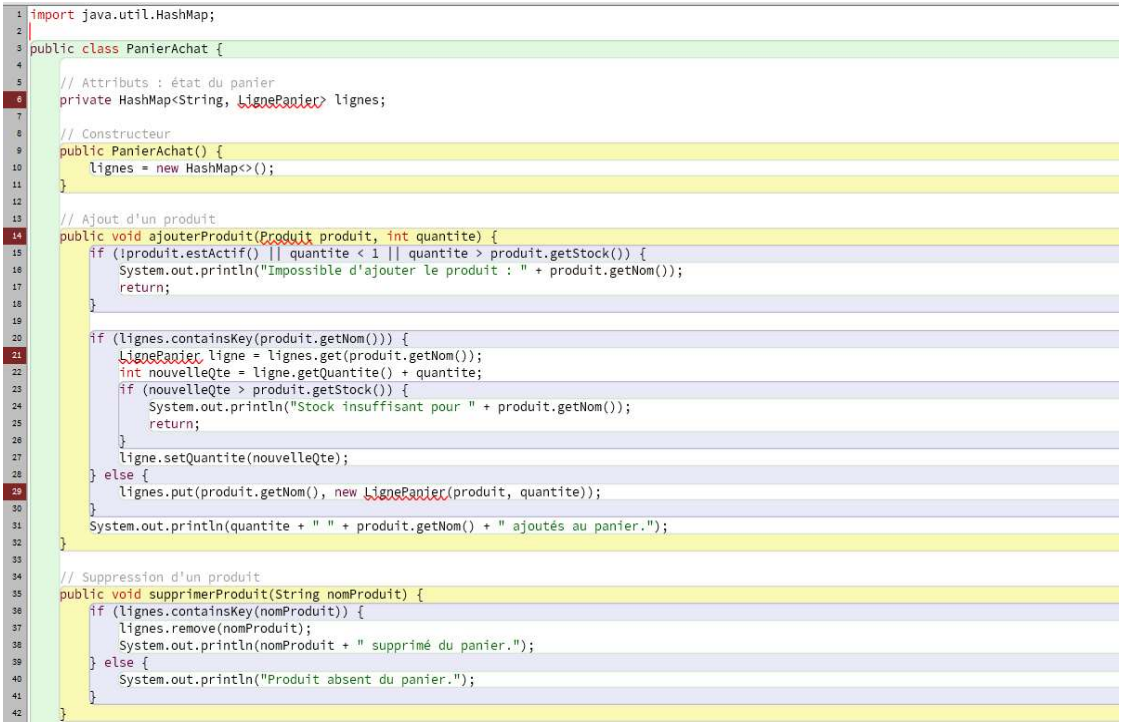
Tutoriel PanierAchat



```java
import java.util.HashMap;

public class PanierAchat {

    // Attributs : état du panier
    private HashMap<String, LignePanier> lignes;

    // Constructeur
    public PanierAchat() {
        lignes = new HashMap<>();
    }

    // Ajout d'un produit
    public void ajouterProduit(Produit produit, int quantite) {
        if (!produit.estActif() || quantite < 1 || quantite > produit.getStock()) {
            System.out.println("Impossible d'ajouter le produit : " + produit.getNom());
            return;
        }

        if (lignes.containsKey(produit.getNom())) {
            LignePanier ligne = lignes.get(produit.getNom());
            int nouvelleQte = ligne.getQuantite() + quantite;
            if (nouvelleQte > produit.getStock()) {
                System.out.println("Stock insuffisant pour " + produit.getNom());
                return;
            }
            ligne.setQuantite(nouvelleQte);
        } else {
            lignes.put(produit.getNom(), new LignePanier(produit, quantite));
        }
        System.out.println(quantite + " " + produit.getNom() + " ajoutés au panier.");
    }

    // Suppression d'un produit
    public void supprimerProduit(String nomProduit) {
        if (lignes.containsKey(nomProduit)) {
            lignes.remove(nomProduit);
            System.out.println(nomProduit + " supprimé du panier.");
        } else {
            System.out.println("Produit absent du panier.");
        }
    }
```

```java
    }

    // Calcul du total
    public double calculerTotal() {
        double total = 0;
        for (LignePanier ligne : lignes.values()) {
            total += ligne.getSousTotal();
        }
        return total;
    }

    // Affichage du panier
    public void afficherPanier() {
        if (lignes.isEmpty()) {
            System.out.println("Panier vide.");
            return;
        }
        for (LignePanier ligne : lignes.values()) {
            System.out.println(ligne);
        }
        System.out.println("Total général : " + calculerTotal());
    }

    // Getter pour tests unitaires
    public HashMap<String, LignePanier> getLignes() {
        return lignes;
    }
}
```

1. Création de la classe Fétiche

La classe est déclarée sous le nom PanierAchat. Elle sert à centraliser la gestion des produits sélectionnés par un utilisateur avant l'achat.

Définition des attributs

- **Structure de données** : Le panier possède un attribut unique nommé lignes.
- **Type utilisé** : Il s'agit d'une HashMap<String, LignePanier>.
- **Rôle** : La String stocke le nom du produit (clé) et LignePanier stocke l'objet produit et sa quantité (valeur).

Ajout des méthodes

- **ajouterProduit** : Vérifie si le produit est actif et si le stock est suffisant. Si le produit est déjà présent, elle met à jour la quantité, sinon elle crée une nouvelle ligne.
- **supprimerProduit** : Retire un article du panier en utilisant son nom pour le supprimer de la HashMap.
- **calculerTotal** : Parcourt toutes les lignes pour additionner les sous-totaux de chaque article.
- **afficherPanier** : Affiche le détail de chaque ligne et le montant total général.
- **getLignes** : Un "getter" permettant d'accéder aux données pour les tests unitaires.

Remarque : Il est normal qu'une **erreur de compilation** apparaisse à ce stade, car les classes dépendantes Produit et LignePanier n'ont pas encore été créées.

```java
public class Produit {

    private String nom;
    private double prix;
    private int stock;
    private boolean actif;

    public Produit(String nom, double prix, int stock, boolean actif) {
        this.nom = nom;
        this.prix = prix;
        this.stock = stock;
        this.actif = actif;
    }

    public String getNom() {
        return nom;
    }

    public double getPrix() {
        return prix;
    }

    public int getStock() {
        return stock;
    }

    public boolean estActif() {
        return actif;
    }

    public void retirerStock(int quantite) {
        if (quantite <= stock) {
            stock -= quantite;
        }
    }
}
```

2. Création de la classe Produit

Cette classe définit les caractéristiques de base d'un article en vente dans le système.

Définition des attributs

- Nom (String) pour désigner l'article.
- Prix (double) pour la valeur monétaire.
- Stock (int) pour la quantité disponible en magasin et actif (boolean) pour savoir si le produit est disponible à la vente.

Ajout des méthodes

- **Constructeur** : Initialise un produit avec toutes ses informations dès sa création.
- **Accesseurs (Getters)** : Permettent de consulter le nom, le prix et le stock (ex: getNom, getPrix).
- **Méthode métier (retirerStock)** : Permet de décréter le stock lors d'un achat, après avoir vérifié que la quantité demandée est disponible.

```java
public class LignePanier {

    private Produit produit;
    private int quantite;

    public LignePanier(Produit produit, int quantite) {
        this.produit = produit;
        this.quantite = quantite;
    }

    public int getQuantite() {
        return quantite;
    }

    public void setQuantite(int quantite) {
        this.quantite = quantite;
    }

    public double getSousTotal() {
        return produit.getPrix() * quantite;
    }

    @Override
    public String toString() {
        return produit.getNom() + " | Prix: " + produit.getPrix() + " | Qté: " + quantite + " | Sous-total: " + getSous
    }
}
```

3. Création de la classe LignePanier

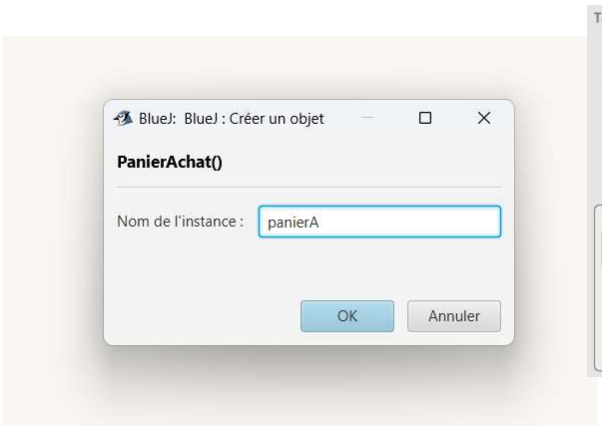
Cette classe sert d'intermédiaire. Elle représente une ligne spécifique dans le panier

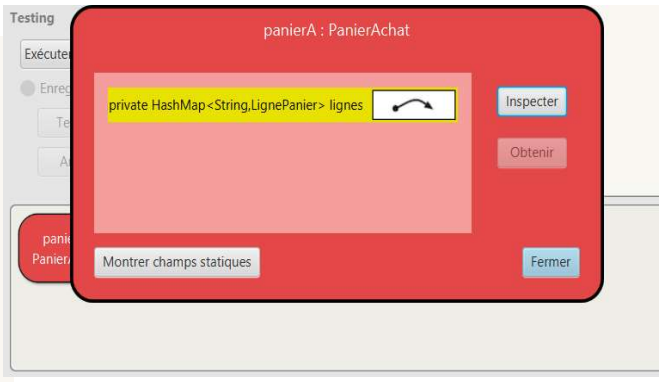
Définition des attributs

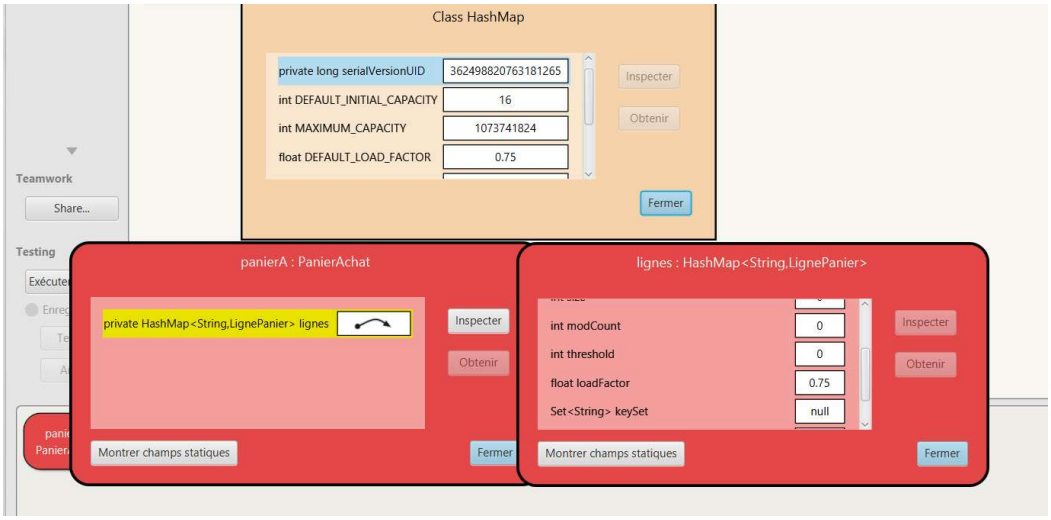
- Un attribut de type Produit. C'est une **association** : la ligne connaît le produit auquel elle fait référence.
- **Quantité** : Un entier (quantite) pour savoir combien d'unités de ce produit l'utilisateur veut acheter.

Ajout des méthodes

- **Constructeur** : Crée la ligne en associant un objet Produit existant à une quantité.
- **Calcul (getSousTotal)** : Multiplie le prix du produit associé par la quantité choisie. C'est ici que la logique financière de la ligne est gérée.
- **Représentation (toString)** : Redéfinition de la méthode standard pour afficher proprement la ligne (Nom | Prix | Qté | Sous-total) lors de l'affichage du panier.







4. Création et Inspection du Panier dans BlueJ

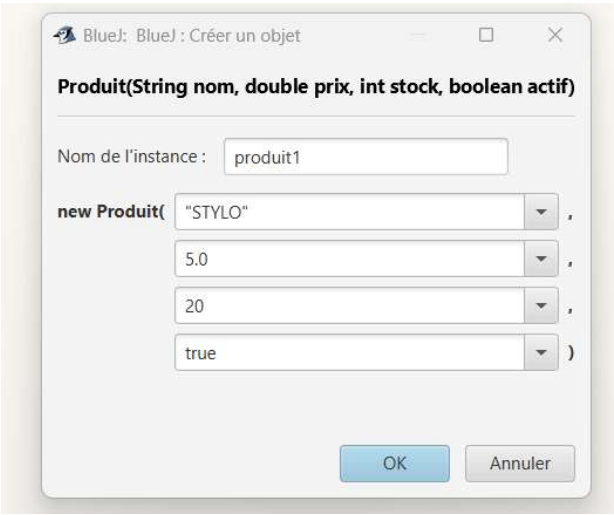
Une fois le code compilé, ces images montrent comment manipuler l'objet en temps réel :

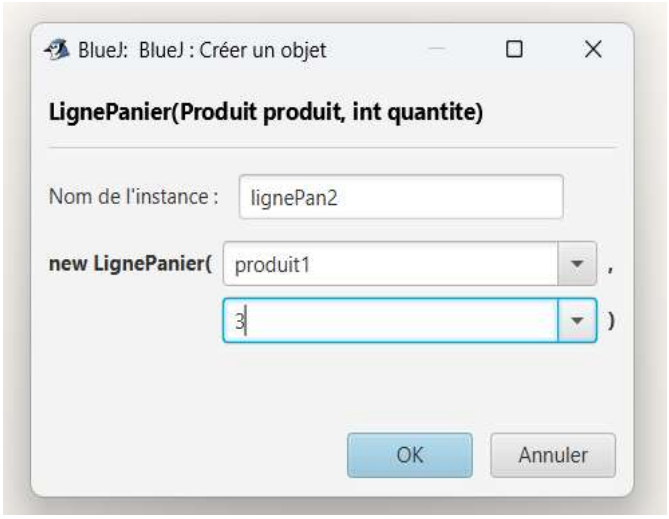
Création de l'instance

- **Action** : On instancie la classe en faisant un clic droit sur PanierAchat.
- **Nom de l'instance** : Dans la fenêtre "Créer un objet", on donne le nom "**panierA**" à notre nouveau panier.
- **Résultat** : Cela appelle le **constructeur** qui initialise la HashMap vide pour accueillir les futurs produits.

Inspection de l'objet

- **Visualisation** : En ouvrant l'inspecteur d'objet sur "**panierA**", on peut voir son état interne.
- **Attribut affiché** : On retrouve l'attribut privé lignes de type HashMap<String, LignePanier>.
- **État initial** : La flèche à côté de l'attribut indique que l'objet lignes est bien créé en mémoire mais qu'il est actuellement vide.





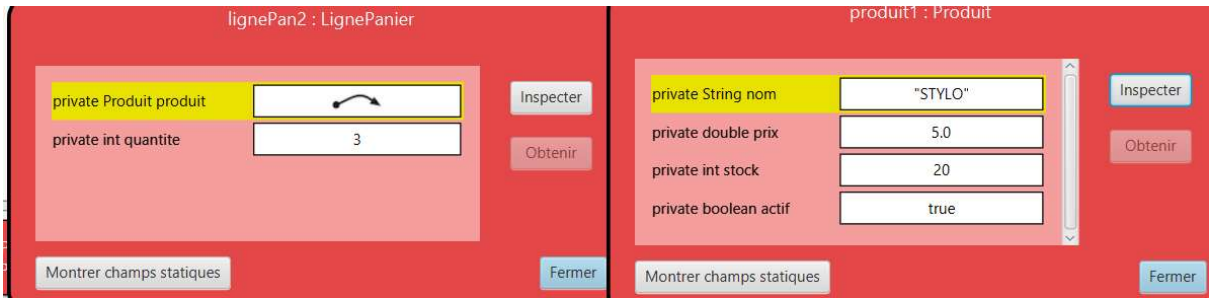
5.Création d'un Produit

- On instancie la classe Produit en remplissant ses attributs via le constructeur.
- **Paramètres saisis** :
  - **Nom de l'instance** : produit1.
  - **Détails** : On crée un "STYLO" au prix de 5.0, avec un stock initial de 20 et l'état true (actif).
- **Rôle** : Cet objet devient la référence qui sera utilisée pour remplir le panier.

6. Création d'une Ligne de Panier

On crée une instance de LignePanier pour lier un produit à une quantité spécifique.

- **Configuration** :
  - **Nom de l'instance** : lignePan2.
  - **Lien d'objet** : Au lieu de taper du texte, on sélectionne l'objet produit1 précédemment créé.
  - **Quantité** : On définit une quantité de 3 pour cet achat



7. Inspecter l'instance lignePan2 de la classe LignePanier

En utilisant la fonctionnalité **Inspecter**, on visualise les **attributs internes** de l'instance lignePan2 de la classe LignePanier.

Lors de l'inspection :

- L'attribut produit correspond à une **référence vers une instance de la classe Produit**
- L'attribut quantite a pour valeur **3**

8. Inspecter l'instance produit1 de la classe Produit

En cliquant sur **Inspecter**, on accède aux détails de l'instance produit1 de la classe Produit.

👉 L'instance inspectée correspond à **un produit actif, disponible en stock et prêt à la vente**.

```java
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class PanierAchatTest {

    @Test
    public void testAjouterProduit() {
        PanierAchat panier = new PanierAchat();
        Produit livre = new Produit("Livre", 15.0, 5, true);
        panier.ajouterProduit(livre, 2);

        assertEquals(1, panier.getLignes().size());
        assertEquals(2, panier.getLignes().get("Livre").getQuantite());
        assertEquals(30.0, panier.calculerTotal());
    }

    @Test
    public void testSuppressionProduit() {
        PanierAchat panier = new PanierAchat();
        Produit livre = new Produit("Livre", 15.0, 5, true);
        panier.ajouterProduit(livre, 2);
        panier.supprimerProduit("Livre");

        assertEquals(0, panier.getLignes().size());
    }

    @Test
    public void testStockInsuffisant() {
        PanierAchat panier = new PanierAchat();
        Produit souris = new Produit("Souris", 50.0, 1, true);
        panier.ajouterProduit(souris, 2); // > stock
        assertEquals(0, panier.getLignes().size());
    }
}
```

9. Création des Tests Unitaires avec JUnit

Cette étape est essentielle pour valider la fiabilité de votre application. On utilise des méthodes de "test" pour comparer le résultat obtenu par le code avec le résultat attendu.

Configuration des tests

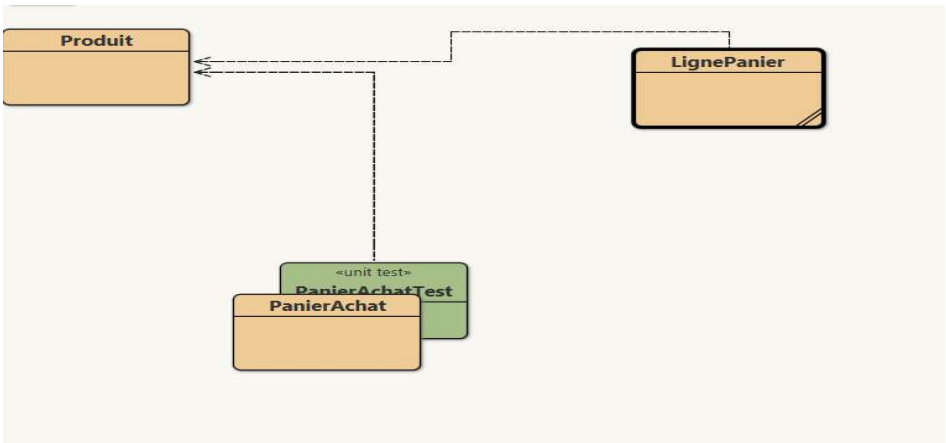
- **Importations** : On importe les outils JUnit (Assertions et Test) pour pouvoir effectuer les vérifications[1].
- **Structure** : Chaque scénario de test est marqué par l'annotation @Test.

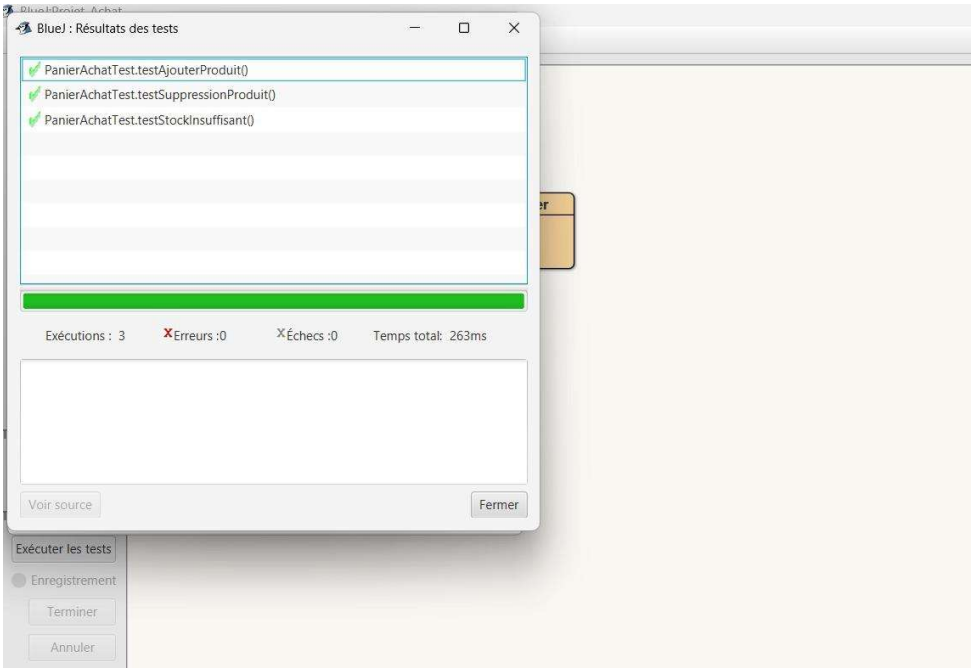
Les scénarios de test (Méthodes)

- **Test d'Ajout (testAjouterProduit)** :
  - On crée un panier et un produit (un "Livre" à 15.0€)[3].
  - On en ajoute 2 au panier[4].
  - **Vérifications** : On s'assure qu'il y a bien 1 ligne dans le panier, que la quantité est de 2, et que le total est bien de 30.0€ (15×2)[5].
- **Test de Suppression (testSuppressionProduit)** :
  - On ajoute un produit puis on le supprime par son nom[6].
  - **Vérification** : On vérifie que la taille du panier revient à **0**[7].
- **Test de Sécurité (testStockInsuffisant)** :
  - On tente d'ajouter 2 souris alors qu'il n'y en a qu'**une** en stock[8].
  - **Vérification** : On vérifie que l'ajout a été refusé et que le panier est resté vide (taille 0).

```
64
65      // Getter pour tests unitaires
66      public HashMap<String, LignePanier> getLignes() {
67          return lignes;
68      }
69  }
70
```

Ajout de getter dans PanierAchat pour les tests unitaires (**getLignes** : Un "getter" permettant d'accéder aux données pour les tests unitaires.)





11. Validation et Tests dans BlueJ

Une fois ces classes créées, vous pouvez valider leur fonctionnement comme illustré dans les images du document.

Le code de test PanierAchatTest permet de vérifier les scénarios critiques :

- **testAjouterProduit** : Vérifie que l'ajout fonctionne et que le total est correct.

- **testStockInsuffisant** : S'assure qu'on ne peut pas commander plus que le stock disponible.

Résultat final : L'image des résultats de tests montre une **barre verte,** ce qui confirme que toutes les méthodes (ajout, suppression, gestion du stock) fonctionnent parfaitement