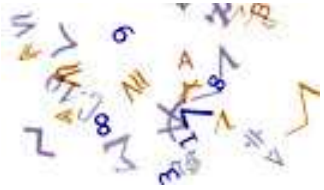


L'environnement de développement

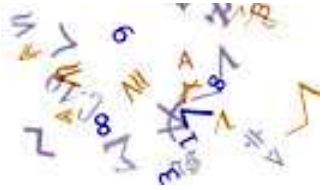
Eclipse Galileo



Origine du projet Eclipse

- Eclipse a été créé par OTI (filiale d'IBM), responsable des environnements de développement intégrés (IDE)
 - Issu de l'environnement VisualAge (Java/SmallTalk)
- Historique

Avril 1999	début d'Eclipse, interne à OTI/IBM
Octobre 2001	Première version stable Eclipse 1.0
Novembre 2001	IBM «donne» Eclipse sous licence OpenSource
Juin 2002	Eclipse 2.0
Juin 2004	Eclipse 3.0
...	...
Juin 2009	Eclipse 3.5 « Galileo »



Qu'est ce qu'Eclipse ?

- Eclipse est :
 - Une plate-forme universelle pour des environnements de développement intégrés
 - Fondée sur une architecture ouverte et extensible

Environnement de développement plug-in

Outils de développement Java

Plate-forme Eclipse

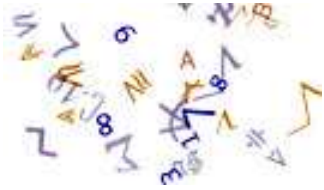
Machine virtuelle Java2

Perspective Java



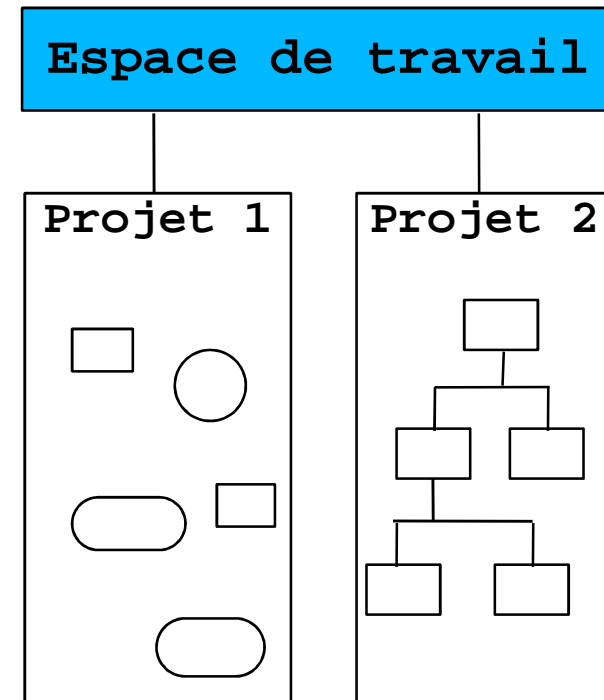
The screenshot shows the Eclipse IDE in the Java Perspective. The interface is annotated with red circles and text labels:

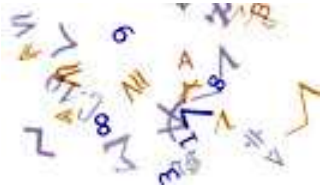
- Vue de l'espace de travail**: A red circle highlights the Package Explorer on the left side of the IDE.
- Editeur**: A large red circle highlights the central editor window displaying the source code of `Server.java`.
- Vue d'une classe**: A red circle highlights the Outline view on the right side, showing the class structure of `Server`.
- Autres vues**: A red circle highlights the bottom toolbar containing views like `Tasks`, `Classic Search`, `Console`, `Properties`, and `Java Beans`.
- Vue sur la console**: A red circle highlights the Console view at the bottom of the IDE.



L'espace de travail

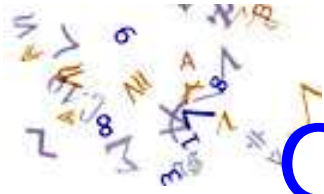
- L'espace de travail (*workspace*)
 - contient tous les fichiers manipulés
 - autorise la création, sauvegarde, modification ou suppression de ces ressources
 - est organisé en un ensemble de projets
 - Correspond à un répertoire précis du disque





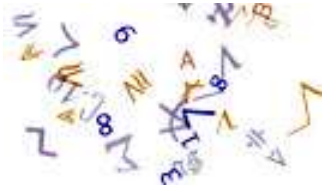
Projet

- Un projet est un regroupement de ressources (fichiers, répertoires, projets)
- Un projet peut être :
 - créé
 - ouvert : il est pris en compte
 - fermé : il est ignoré
 - détruit : avec ou sans ses ressources



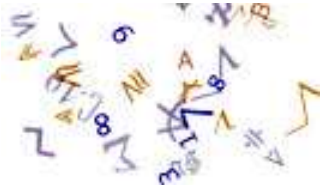
Outils de développement Java

- **Compilateur**
 - compilation incrémentale de tous les projets ouverts (*build*)
- **Exécution**
 - différents types (application, applet, bean, ...)
 - configurable (classe exécutable, classpath, ...)
- **Débugueur (exemple)**
 - exécution pas-à-pas ou avec des points d'arrêt (*breakpoint*)
 - visualisation de la valeur des variables



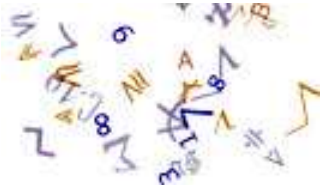
Assistance de l'éditeur Java

- Complète automatiquement des noms de méthodes
- Détecte certaines erreurs et avertissements avant la compilation
- Propose des corrections d'erreur
- Ajoute les importations nécessaires
- Outils de travail coopératif intégré (CVS)



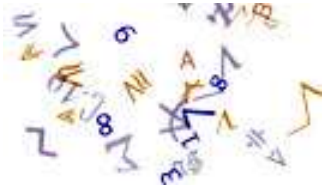
Archives JAR

- Java Archive
- Extension du format ZIP
- Avantages
 - ◆ Archivage
 - ◆ Production d'exécutables portables (à condition d'avoir une JVM)
 - Alternative au classique .exe
 - Lancement aussi simple (double-clic, ou ligne de commande)
- Partage, ré-utilisation

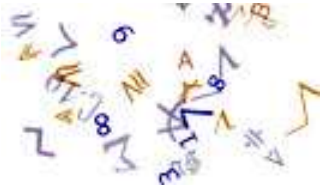


Plug-ins

- Un plug-in étend la plate-forme Eclipse pour certains types de développement
 - Téléchargeables et à installer par le menu
 - *Help -> Install New Software...*
 - Par exemple: <http://cloudgarden1.com/update-site>
 - *Jigloo GUI Builder -> Install...*
 - Un plug-in peut en nécessiter d'autres
 - Possibilité de développer de nouveaux plug-ins
 - Un site répertorie les principaux plug-ins disponibles :
<http://www.eclipseplugincentral.com/>



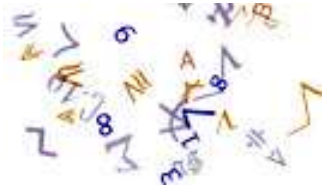
Le plug-in **Jigloo**



Le plug-in Jigloo

- Outil de construction d'interfaces graphiques et de génération automatique de code
- Un éditeur permet de visualiser l'apparence d'une interface graphique pendant sa construction
- Construction par « drag and drop » d'éléments graphiques (containers, components)
- Ecran d'édition des propriétés des composants (e.g. dimensions, couleur, texte d'un bouton, ...)

Aperçu de Jigloo



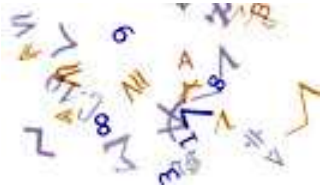
Editeur graphique

```

jLabel1 = new JLabel();
jPanel1.add(jLabel1, BorderLayout.NORTH);
getContentPane().add(getJPanel1(), BorderLayout.CENTER);
jPanel1.setBorder(BorderFactory.createCompoundBorder(
    BorderFactory.createBevelBorder(BevelBorder.RAISED),
    BorderFactory.createEmptyBorder(20, 20, 20, 20)));
jLabel1.setText("Entrez votre code");
jLabel1.setFont(new Font("Tahoma", Font.PLAIN, 14));
jLabel1.setHorizontalAlignment(SwingConstants.CENTER);
    
```

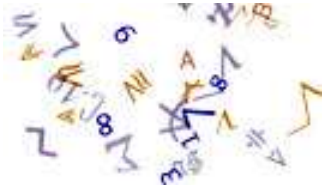
Properties	Value	Layout	Value	Event Name	Value
Background	[212, 208, 200]	Constraints	Border	AncestorListener	<none>
border	No Border	Layout	Absolute	ComponentListener	<none>
enabled	<input checked="" type="checkbox"/> true			ContainerListener	<none>
focusTraversalPolicy	[]			FocusListener	<none>
font*	Tahoma, 14, Bold			HierarchyBoundsListener	<none>
foreground	[0, 0, 0]			HierarchyListener	<none>
icon	No Icon			InputMethodListener	<none>
locale	français (France)			KeyListener	<none>
opaque	<input type="checkbox"/> false			MouseListener	<none>
preferredSize	[0, 0]			MouseMotionListener	<none>
size	[441, 17]			MouseWheelListener	<none>
text*	Entrez votre code			Runnable	<none>
toolTipText				TextAreaListener	<none>
Expert				TextAreaListener	<none>
Hidden				TextAreaListener	<none>

Propriétés d'un composant

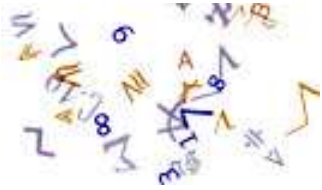


Exemples

- Création d'un panel avec champ de texte et bouton
(exemple)
- Gestion d'événement sur le bouton
(exemple)

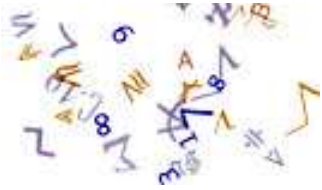


Tests unitaires avec **Junit 4**



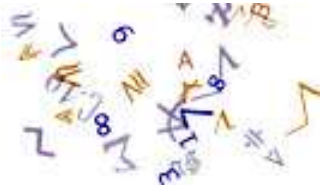
Éléments du framework

- Instructions de test **classe Assert**
- Méthodes de test **@Test**
- Méthodes **fail()**
- Méthodes d'initialisation **@Before**
- Méthodes de finalisation **@After**
- Test Case



JUnit : assert

- Méthodes static de la classe *org.junit.Assert*
 - assertEquals
 - assertTrue, assertFalse
 - assertNull, assertNotNull
 - assertEquals, assertEquals
- Paramètres d'invocation
 - expected, actual
 - expected, actual, delta
 - condition
 - ... + message

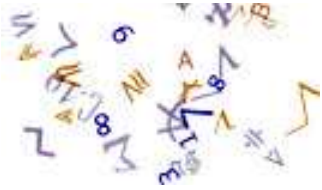


Méthodes de test

- Méthode qui exécute un test unitaire
- Convention de nommage test[méthode à tester]()
- Utilise l'annotation @Test

@Test

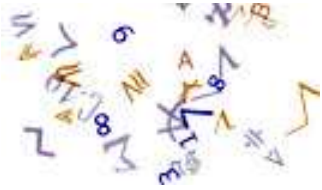
```
public void testSetMontant() {  
    compte.setMontant(100000.0);  
    assertEquals(compte.getMontant(), 100000.0);  
}
```

Méthode fail()

- Méthode qui provoque l'échec d'un test

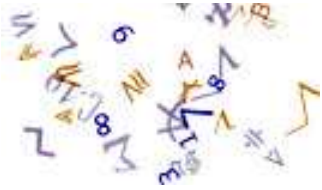
```
@Test
public void testSetMontant() {
    if (compte == null)
        fail("Erreur d'initialisation du test");
    compte.setMontant(100000.0);
    assertEquals(compte.getMontant(), 100000.0);
}
```



Méthode d'initialisation

- Méthodes d'initialisation avant chaque test
- `setUp()` est une convention de nommage
- L'annotation `@Before` est utilisée

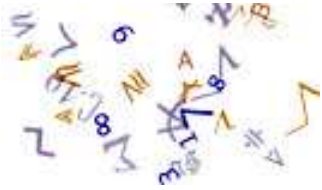
```
@Before  
public void setUp() {  
    compte = new Compte();  
}
```



Méthode de finalisation

- Méthodes de finalisation après chaque test
- `tearDown()` est une convention de nommage
- L'annotation `@After` est utilisée

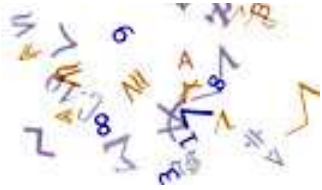
```
@After  
public void tearDown() {  
    declaration = null;  
}
```



Test case

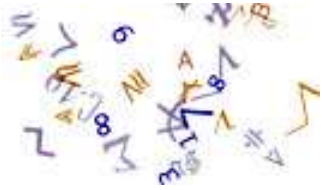
- Classe qui regroupe plusieurs méthodes de tests unitaires
- Une classe de test par classe à tester
- Convention de nommage : [*Classe à tester*]Test
- Pour faciliter l'emploi des méthodes static de *org.junit.Assert* on utilise

```
import static org.junit.Assert.*;
```



Test case (2)

```
public class CompteTest {  
    private Compte compte;  
    @Before public void setUp() {  
        compte = new Compte();  
        compte.setMontant(5000);  
    }  
    @After public void tearDown() {  
        compte = null;  
    }  
    @Test public void testEstDebiteur() {  
        assertFalse(compte.estDebiteur());  
    }  
}
```



Suite de test cases

- Plusieurs test cases peuvent être lancés à la suite dans une classe décrivant cette suite
- l'annotation

@RunWith(suite.class)

indiquée avant la déclaration de la classe le permet

- l'annotation

**@SuiteClasses(value={CompteTest.class,
ClientTest.class})**

indiquée juste après donne les test cases à exécuter

