Desktop Intégration

Rémi Forax

Intégration au bureau

- L'intégration de l'application au bureau de l'utilisateur se fait en combinant plusieurs techniques
  - L'écran d'annonce de l'application (**SplashScreen**)
  - Définition des Icônes de l'application
  - Le contrôle des icônes dans la zone de notification (**SystemTray**)
  - La possibilité de demander au bureau d'ouvrir un fichier avec l'application par défaut, de visualiser un fichier dans le *browser web*, etc. (**Desktop**)
  - La possibilité de prendre le contrôle du bureau pour générer de faux évènements souris/clavier ou faire des copies d'écrans (**Robot**)

SplashScreen

- Il est possible de spécifier une image de démarrage de l'application
- Cette image est chargée **avant** toutes les classes de la machine virtuelle

```
java -splash:splash.jpg fr.umlv.ui.splash.SplashTest
```

SplashScreen & Jar

- Il est possible de spécifier le *splash screen* dans le **manifest** d'un fichier jar

```
Manifest-Version: 1.0
Main-Class: Test
SplashScreen-Image: splash.jpg
```

- Le *splash screen* est alors affiché lors de l'exécution du jar

```
java -jar splash-test.jar
```

SplashScreen

- Pour obtenir le splash screen, on utilise la *method factory* **SplashScreen.getSplashScreen();**

- Le splash screen disparaît :
  - Lors de l'affichage de la première fenêtre

```
frame.setVisible(true);
```

  - Lors d'un appel explicite à **close()**

```
public static void main(String[] args) throws InterruptedException {
  final SplashScreen splash=SplashScreen.getSplashScreen();
  Thread.sleep(1000);

  splash.close();
}
```

Mise à jour dynamique

- Il est possible de dessiner sur le splash screen
- La méthode **Graphics createGraphics()** renvoie le context graphique correspondant à une image transparente par dessus l'image utilisée pour le splash screen

- Il faut alors après les modification effectuées demander la mise à jour de l'image transparent avec la méthode **update()**

Mise à jour dynamique

```
private static Image loadImage(String filename) {
  return new ImageIcon(SplashTest.class.getResource(filename)).getImage();
}
public static void main(String[] args) throws InterruptedException {
  final SplashScreen splash=SplashScreen.getSplashScreen();
  Image[] images=new Image[]{
    loadImage("home.png"),
    loadImage("mona.png"),
    loadImage("stop.png")
  };
  Graphics2D graphics=splash.createGraphics();
  int baseline=splash.getSize().height-70;
  for(int i=0;i<images.length;i++) {
    // chargement du module i
    graphics.drawImage(images[i],20+i*70,baseline,48,48,null);
    splash.update();
  }
  graphics.dispose();
  splash.close();
}
```

Utiliser une barre de progression

- On souhaite afficher une barre de progression sur le *splash screen*
- Il n'est pas possible de faire un **add**() d'un **JProgressBar** car le *splash screen* n'est pas un container

- Il faut demander l'affichage du **JProgressBar** directement dans le **graphics** de l'image transparente située au dessus du *splash screen*

Utiliser une barre de progression

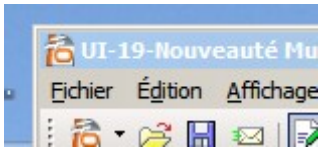
- Pour afficher la barre de progression au bonne endroit, il faut translater le graphics.
- Un composants se dessine toujours en 0,0

```
public static void main(String[] args) throws InterruptedException {
  final SplashScreen splash=SplashScreen.getSplashScreen();
  Dimension size=splash.getSize();

  JProgressBar bar=new JProgressBar(0,100);
  bar.setOpaque(true);
  bar.setSize(size.width-20,bar.getPreferredSize().height);
  bar.setVisible(true);

  Graphics2D graphics=splash.createGraphics();
  graphics.translate(10, size.height-30);
  for(int i=0;i<100;i++) {
    bar.setValue(i);
    bar.paint(graphics);
    splash.update();
    Thread.sleep(100);
  }

  graphics.dispose();
  splash.close();
}
```

Icônes de l'application

- Une application possède différents icônes :
  - Icône de la fenêtre
  - Icône de l'application réduite
  - Icône dans la barre des tâches
  - Icône lors du changement d'application (ALT-TAB)
  - etc.

- Ces icônes ont des tailles différentes dépendant de la plateforme



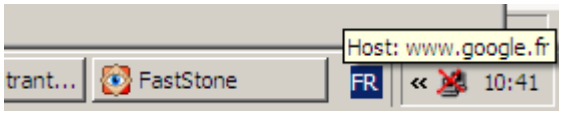
Icônes de l'application

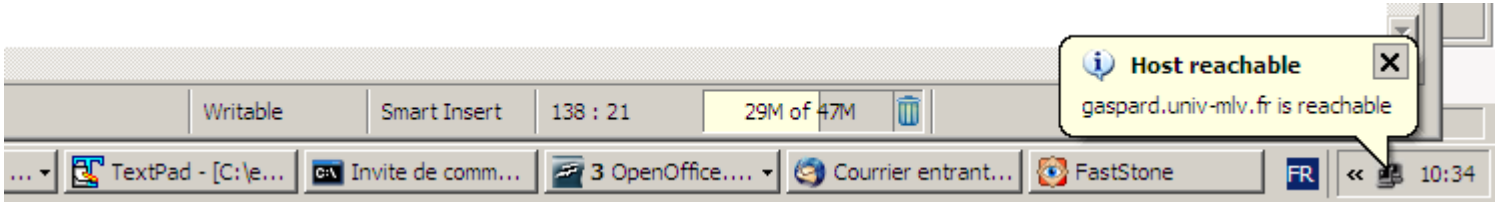
- Permet de spécifier une liste d'icônes
  window.setIconImages(List<? extends Image> icons)
- Java choisi alors l'icône le plus adapté en fonction du contexte et de la taille demandée par l'OS.

```
public static void main(String[] args) {
  Image icon16_16=new ImageIcon(
    IconsFrame.class.getResource("stop16x16.gif")).getImage();
  Image icon32_32=new ImageIcon(
    IconsFrame.class.getResource("stop32x32.gif")).getImage();
  Image icon48_48=new ImageIcon(
    IconsFrame.class.getResource("stop48x48.png")).getImage();

  JFrame frame=new JFrame();
  frame.setIconImages(Arrays.asList(icon16_16,icon32_32,icon48_48));
  frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  frame.setSize(400,300);
  frame.setVisible(true);
}
```

System Tray

- **java.awt.SystemTray** contrôle la zone de notification de la barre des tâches
  - Le bureau possède une zone de notification supporté boolean SystemTray.isSupported()
  - Obtenir la zone de notification (*singleton*) : SystemTray SystemTray.getSystemTray()

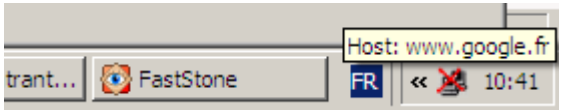




System Tray

- L'application peut alors gérer les icônes de la zone de notification
  - Ajouter/retirer des icônes
    add/removeTrayIcon(TrayIcon)
  - Obtenir les icônes présents
    TrayIcon[] getTrayIcons()

- Il est possible de connaître la taille des icônes
  - Taille en pixels des icônes
    Dimension getTrayIconSize()



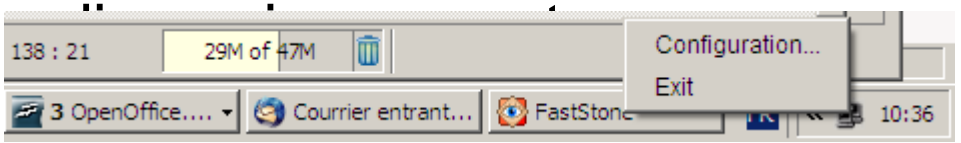
System Tray

- La classe **SystemTray** possède un mécanisme générique de *listener* pour être averti des changements
  - Enregistrer/Dé-enregistrer un *listener* add/removePropertyChangeListener(String propertyName, PropertyChangeListener listener)
  - Obtenir les *listeners* PropertyChangeListener[] getPropertyChangeListeners(String propertyName)
  - ajouts ou suppressions d'icônes (propriété: **`"trayIcons"`**)

Tray Icon

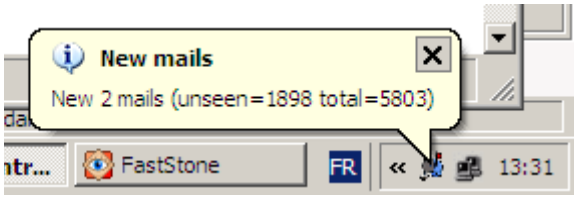
- Un icône de la zone de notification est représentée par la classe **java.awt.TrayIcon**
- **TrayIcon(image,tooltip,popup)**
  - Une image (java.awt.Image)
  - Un tooltip (String),
  - Un popup menu (PopupMenu)

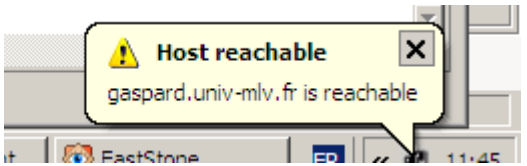
- Il est possible de demander un automatique de l'image
  - get/setImageAutoSize(boolean)

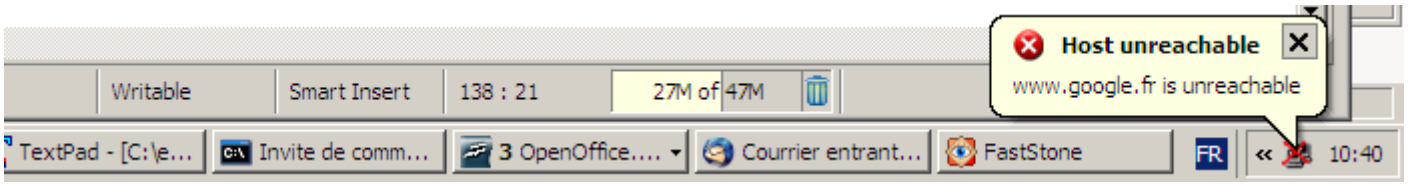


Tray Icon

- Il est possible d'afficher des messages (balloon message) pour notifier l'utilisateur
  - **void displayMessage(String caption, String text, TrayIcon.MessageType messageType)**
- Les types de messages sont :
  - message simple : NONE
  - message d'information : INFO
  - message d'attention : WARNING
  - message d'erreur : ERROR







Tray Icon

- Un icône de la zone de notification possède trois types de *listeners*
  - La sélection de l'icône par la souris ou le clavier add/removeActionListener()
  - Les évènements souris add/removeMouseListener()
  - Les évènements souris impliquant un déplacement add/removeMouseMotionListener()

Test accessibilité d'une machine

- Enregistrer une application dans la zone de notification qui indique si une machine est atteignable

```
public static void main(String[] args) throws AWTException {
  final SystemTray tray=SystemTray.getSystemTray();
  Dimension traySize=tray.getTrayIconSize();

  final Image trayOk=loadImage("tray_ok.gif",traySize);
  final Image trayError=loadImage("tray_error.gif",traySize);

  String tooltip=...
  PopupMenu popup=new PopupMenu();
  final TrayIcon trayIcon=new TrayIcon(trayError,tooltip,popup);
  tray.add(trayIcon);
  final ReachableTester tester=new ReachableTester(...);

  MenuItem conf=new MenuItem("Configuration...");
  conf.addActionListener(new ActionListener() {
    ...
  });
  popup.add(conf);
  MenuItem exit=new MenuItem("Exit");
  ...
  popup.add(exit);
```

Testeur d'accessibilité

- Le testeur d'existence utilise InetAdress.isReachable()
- Un scheduledExecutorService permet d'effectuer le test périodiquement

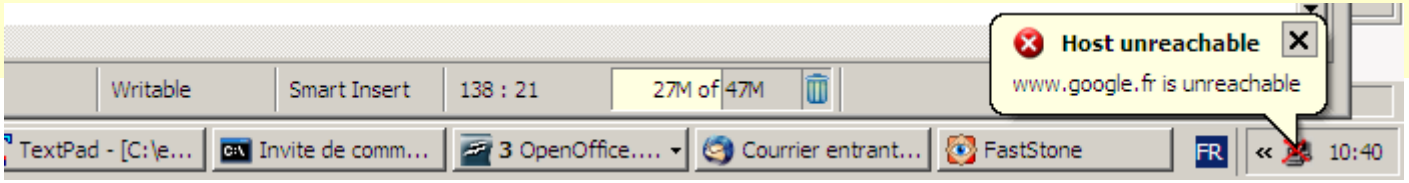
```
private interface ReachableListener {
  public void hostUnreachable(InetAddress address);
  public void hostReachable(InetAddress address);
}
private static class ReachableTester {
  public ReachableTester(InetAddress address,ReachableListener listener) {
    this.address=address;
    this.listener=listener;
    service=Executors.newSingleThreadScheduledExecutor();
    service.scheduleWithFixedDelay(runnable,3,3,TimeUnit.SECONDS);
  }
  public void shutdownNow() {
    service.shutdownNow();
  }
  public void setAddress(InetAddress address) {
    this.address = address;
  }
  private final Runnable runnable=new Runnable() {
    ...
  };
  volatile InetAddress address;
  final ReachableListener listener;
  private final ScheduledExecutorService service;
```

Testeur d'accessibilité (suite)

```
private static class ReachableTester {
  ...
  private final Runnable runnable=new Runnable() {
    public void run() {
      // volatile read
      InetAddress address=ReachableTester.this.address;

      boolean isReachable=false;
      if (address!=null) {
        try {
          isReachable=address.isReachable(3000);
        } catch(IOException e) {
          // do nothing, isReachable==false
        }
      }
      if (isReachable!=this.isReachable) {
        if (isReachable)
          listener.hostReachable(address);
        else
          listener.hostUnreachable(address);
        this.isReachable=isReachable;
      }
    }
    private boolean isReachable=false;
  };
  volatile InetAddress address;
  final ReachableListener listener;
  private final ScheduledExecutorService service;
}
```

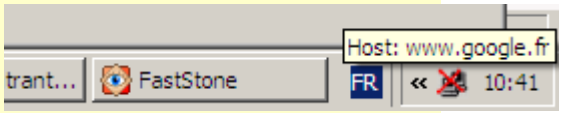
Testeur d'accessibilité (suite)

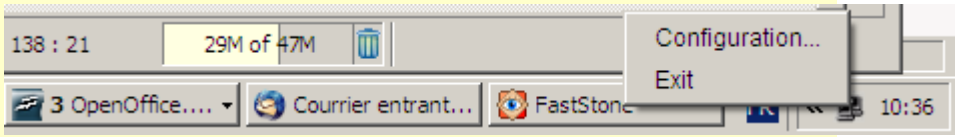
```
String tooltip;
InetAddress address;
try {
  address = InetAddress.getByName("gaspard.univ-mlv.fr");
  tooltip="Host: "+address.getHostName();
} catch (UnknownHostException e) {
  address=null;
  tooltip="Unknown host";
}
PopupMenu popup=new PopupMenu();
final TrayIcon trayIcon=new TrayIcon(trayError,tooltip,popup);
tray.add(trayIcon);

final ReachableTester tester=new ReachableTester(address,
  new ReachableListener() {
    public void hostReachable(InetAddress address) {
      trayIcon.setImage(trayOk);
      trayIcon.displayMessage("Host reachable",
        address.getHostName()+" is reachable",MessageType.INFO);
    }
    public void hostUnreachable(InetAddress address) {
      trayIcon.setImage(trayError);
      trayIcon.displayMessage("Host unreachable",
        address.getHostName()+" is unreachable",MessageType.ERROR);
    }
  }
);
```



Testeur d'accessibilité (fin)

```
MenuItem conf=new MenuItem("Configuration...");
conf.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent event) {
    String host=JOptionPane.showInputDialog(null,"Enter the host name");
    if (host==null)
      return;

    try {
      InetAddress address=InetAddress.getByName(host);
      tester.setAddress(address);
      trayIcon.setToolTip("Host: "+address.getHostName());
    } catch (UnknownHostException e) {
      JOptionPane.showMessageDialog(null,e.getMessage(),
        "Unknown host",JOptionPane.ERROR_MESSAGE);
    }
  }
});
popup.add(conf);

MenuItem exit=new MenuItem("Exit");
exit.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    tray.remove(trayIcon);
    tester.shutdownNow();
  }
});
popup.add(exit);
```





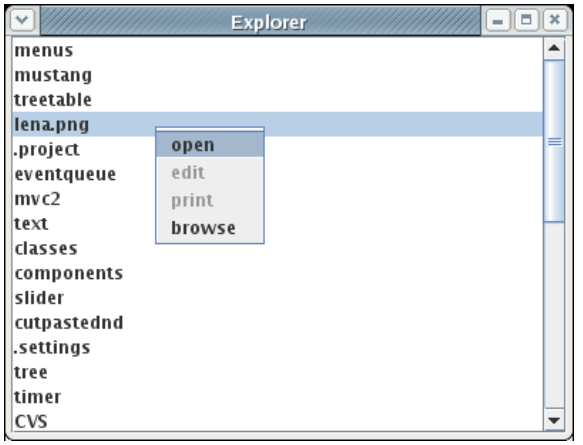
Bureau

- **java.awt.Desktop** permet de :
  - demander pour une fichier d'exécuter l'application par défaut pour des actions prédéfinies (OPEN,EDIT ou PRINT)
  - De demander au *browser* par défaut d'afficher une URI
  - De demander au *mailer* par défaut de composer un mail

- Les actions sont définies par l'énumération **Desktop.Action** :
  OPEN, **EDIT**, **PRINT**, **BROWSE**, **MAIL**.

Bureau

- Permet de savoir si un objet **Desktop** existe :
  - **boolean Desktop.isDesktopSupported()**
- Pour obtenir le bureau, il faut utiliser la *factory* :
  - **Desktop.getDesktop()**



Bureau

- Opérations sur les fichiers utilisant l'application par défaut :
  - Ouvrir un fichier : **open**(File file)
  - Editer un fichier : **edit**(File file)
  - Afficher un fichier : **print**(File file)
- Opération sur des applications particulières :
  - Ouvrir une URI avec le browser : **browse**(URI uri)
  - Ouvrir une fenêtre de composition de mail : **mail**(URI uri)
    avec un URI de la forme :
    mailto:remi@forax.org?subject=Hello&body=mailworld
    (les champs possibles sont "to", "cc", "subject", "body")

Exemple

```
final JList list=new JList(new FileListModel(new File(".")));
list.setCellRenderer(...);

Desktop desktop=Desktop.getDesktop();
Desktop.Action[] desktopActions={OPEN,EDIT,PRINT,BROWSE};
Action[] actions=new Action[desktopActions.length];
final ArrayList<Action> supportedActions=new ArrayList<Action>();
for(int i=0;i<desktopActions.length;i++) {
  Desktop.Action desktopAction=desktopActions[i];
  Action action=createAction(list,desktop,desktopAction);
  action.setEnabled(false);
  if (desktop.isSupported(desktopAction))
    supportedActions.add(action);
  actions[i]=action;
}

list.getSelectionModel().addListSelectionListener(
  new ListSelectionListener() {
    public void valueChanged(ListSelectionEvent e) {
      boolean selected=list.getSelectedIndex()!=-1;
      for(Action action:supportedActions)
        action.setEnabled(selected);
    }
});

JPopupMenu popup=new JPopupMenu();
for(Action action:actions)
  popup.add(action);
list.setComponentPopupMenu(popup);
```

Exemple

```
private static Action createAction(final JList list,
  final Desktop desktop,final Desktop.Action desktopAction) {
  final String name=desktopAction.name().toLowerCase();
  return new AbstractAction(name) {
    public void actionPerformed(ActionEvent event) {
      loop: for(Object value:list.getSelectedValues()) {
        File file;
        try {
          file = ((File)value).getCanonicalFile();
          switch(desktopAction) {
            case OPEN:
              desktop.open(file);
              continue loop;
            case EDIT:
              desktop.edit(file);
              continue loop;
            case PRINT:
              desktop.print(file);
              continue loop;
            case BROWSE:
              desktop.browse(file.toURI());
              continue loop;
            default:
              throw new AssertionError("invalid desktop action "+desktopAction);
          }
        } catch (IOException e) {
          e.printStackTrace();
        } } } };
}
```

Robot

- La classe **java.awt.Robot** permet de prendre le contrôle du bureau pour :
  - Récupérer une image du bureau
  - Envoyer des évènements de façon programmé
- Le robot à deux utilisations principales :
  - Prendre le contrôle d'un terminal graphique à distance
  - Tester de façon automatique les interfaces graphiques

Gestion du robot

- Créer une capture d'écran :
  - BufferedImage createScreenCapture(Rectangle rect)
- Fixer le delay entre deux évènements :
  - get/setAutodelay(int milliseconds)
- Demande d'attendre un certain délai :
  - delay(int milliseconds)
- Indique d'attendre après chaque commande que l'ensemble des évènements soit traité :
  - get/setAutoWaitForIdle(boolean onOff)

Evènements générables

- Génération d'évènement :
  - Déplacement de souris :
    - mouseMove(int x,int y)
  - Boutons de la souris :
    - mousePress(int buttons), mouseRelease(int buttons)
  - Molette de la souris :
    - mouseWheel(int wheelAmt)
  - Touche clavier :
    - keyPress(int keycode), keyRelease(int keycode)

Exemple

- On enregistre l'application dans le *system tray*, un double-clique effectue une copie d'écran

```
final SystemTray tray=SystemTray.getSystemTray();

Image camera=new ImageIcon(
  Snapshot.class.getResource("camera.jpg")).getImage();

PopupMenu popup=new PopupMenu();
final TrayIcon trayIcon=new TrayIcon(camera,"Snapshot",popup);
trayIcon.setImageAutoSize(true);

trayIcon.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent event) {
    ...
  }
});
tray.add(trayIcon);

MenuItem exit=new MenuItem("Exit");
exit.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    tray.remove(trayIcon);
    System.exit(0);
  }
});
}
```

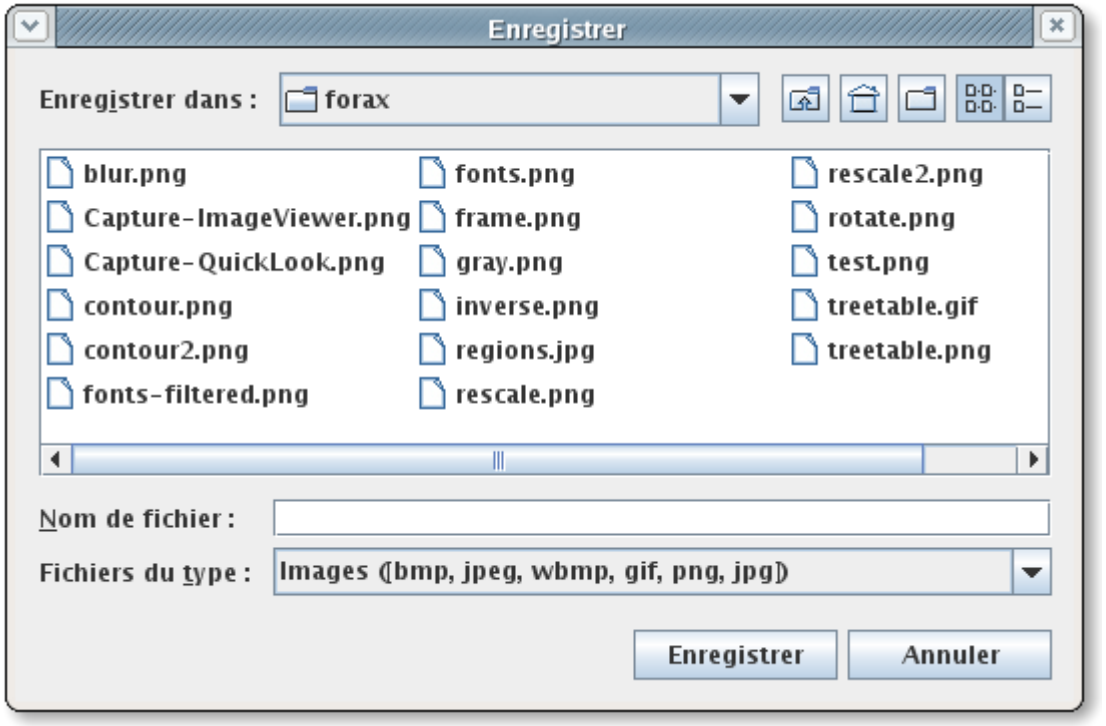
Exemple

- On calcule la taille de l'écran, on fait une copie,
  puis on récupère l'ensemble des suffixes d'images connues
  (png, jpg etc.) et l'on ouvre un *filechooser*

```
GraphicsEnvironment env=GraphicsEnvironment.getLocalGraphicsEnvironment();
Rectangle rect=env.getDefaultScreenDevice().
  getDefaultConfiguration().getBounds();
BufferedImage image;
try {
  image=new Robot().createScreenCapture(rect);
} catch (AWTException e) {
  JOptionPane.showMessageDialog(null,e.getMessage(),
  e.getClass().getName(),JOptionPane.ERROR_MESSAGE);
  return;
}

JFileChooser chooser=new JFileChooser();
chooser.setMultiSelectionEnabled(false);
String[] suffixes=ImageIO.getReaderFileSuffixes();
FileNameExtensionFilter filter=new FileNameExtensionFilter(
  "Images "+Arrays.toString(suffixes),suffixes);
chooser.setFileFilter(filter);
chooser.showSaveDialog(null);
```

Exemple

- On utilise un **JFileChooser** pour demander à l'utilisateur où sauver l'image et suivant quel format.
- Le format est choisi parmi les formats reconnu par **ImageIO**



Exemple

- **ImageIO** sauvegarde dans le format suivant l'extension

```
        ...
        chooser.showSaveDialog(null);
        File file=chooser.getSelectedFile();
        if (file==null)
          return;

        ImageWriter writer=ImageIO.getImageWritersBySuffix(
           getSuffix(file)).next();
        try {
          writer.setOutput(ImageIO.createImageOutputStream(file));
          writer.write(image);
        } catch (IOException e) {
          JOptionPane.showMessageDialog(null,e.getMessage(),
            e.getClass().getName(),JOptionPane.ERROR_MESSAGE);
        }
      }
    });
    ...
}
static String getSuffix(File file) {
  String filename=file.getName();
  int index=filename.lastIndexOf('.');
  if (index==-1)
    return "";
  return filename.substring(index+1);
}
```