Diese Hausaufgabe ist keine Pflicht - mit ihr kann die schlechteste Abgabe, bzw. EINE versäumte oder nicht bestandene Abgabe (weniger als 50 Prozent) ausgeglichen werden.

Die Aufgaben müssen einzeln bearbeitet und abgegeben werden. Die Abgabe muss bis **spätestens Donnerstag 14.02.2013 um 23:59 Uhr** über unser Hausaufgabenabgabesystem **http://seblog.cs.uni-kassel.de/pmws1213/** erfolgen. Die Abgabe ist nur als einzelne *.zip oder *.jar-Datei möglich. Daher müssen alle für eine Abgabe relevanten Daten zu einem solchen Archiv kombiniert werden. Diese Hausaufgabe gibt **36** Punkte.

Hinweise zur Abgabe:

- Die Hausaufgabe ist als exportiertes Eclipse Projekt (*.zip, **nicht** den gesamten Workspace) abzugeben. Dies kann mit Hilfe der Eclipse Export Funktion durchgeführt werden. Ist das Projekt nicht korrekt exportiert, kann es bei der Korrektur nicht berücksichtigt werden (es bietet sich also an, den Import des exportierten Projektes auszuprobieren).

  WICHTIG Benennen Sie ihre Projekte für diese Abgabe nach folgendem Schema:

  ```
  PMWS1213_HA<a>_<Matrikelnummer>,
  ```

  wobei <a> für die aktuelle Hausaufgabe steht. Beispiel:

  ```
  PMWS1213_HA11_12345678
  ```

  Ein falsch benanntes Projekt führt zu **2** Punkten Abzug.

Allgemeines

In dieser Hausaufgabe soll ein einfaches Whiteboard entwickelt werden, um die Grundlagen der Client/Server Programmierung zu erlangen. Hierzu wird zunächst der Server erstellt, danach soll der Client entwickelt werden. Im Anschluss soll für den Client eine grafische Oberfläche erstellt werden die es erlaubt, sich an einem Server anzumelden und auf ein gemeinsam genutztes Whiteboard zu zeichnen bzw. es zu leeren.

Aufgabe 1: Projekt erstellen (2P)

Erstellen Sie ein Eclipse Projekt, welches die folgenden vier Java Packages im `src` Verzeichnis enthält:

- de.uks.pmws1213.whiteboard.client
- de.uks.pmws1213.whiteboard.client.controller
- de.uks.pmws1213.whiteboard.client.gui
- de.uks.pmws1213.whiteboard.server

Aufgabe 2: Server erstellen (10P)

Erstellen Sie eine Klasse `Server` im Package `de.uks.pmws1213.whiteboard.server`. Der Server hat die Aufgabe an einem bestimmten Port auf eingehende Verbindungen zu hören. Der Port soll dem Server als Parameter beim Starten übergeben werden können. (Tipp: Die main-Methode bietet sich hier an.)

Erstellen Sie danach eine Klasse `ConnectionHandler` im gleichen Package, die von der Klasse `java.lang.Thread` erbt. Der Server soll vier Plätze für eingehende Verbindungen anbieten. Für jede beim Server eingehende Verbindung wird ein neuer `ConnectionHandler` erzeugt und gestartet, der zunächst eine Nummer an den zugehörigen Client sendet und fortan Nachrichten des Clients entgegennimmt und dem Server übergibt. Der Server schickt dann die Nachricht an alle Clients weiter.

Die vergebenen Nummern repräsentieren die Farbe mit welcher der Client auf dem Whiteboard zeichnen. Rot: 255, Grün: 65280, Blau: 16711680, Gelb: 65535

Wird die vom `ConnectionHandler` gehaltene Verbindung unterbrochen, soll der Server von nun an keine Nachrichten mehr an diesen Client verschicken.

Hinweis: Sie können sich bei der Implementierung an der PM Vorlesung 12 und 13, sowie an der Übung 12 orientieren.

Aufgabe 3: Client erstellen (5P)

Erstellen Sie eine Klasse `ClientNetworkHandler` im Package `de.uks.pmws1213.whiteboard.client` welche von der Klasse `java.lang.Thread` erbt. Diese soll im Konstruktor den Host (als `String`) sowie den Port übergeben bekommen. Wird der Handler gestartet, nimmt er Verbindung mit dem Server auf, speichert sich die vom Server zugewiesene Farbkodierung und zerlegt eingehende Nachrichten um sie für die grafische Oberfläche aufzubereiten. Dafür benötigt er folgende Methode:

```
public void parseMessage(String msg)
{
  String[] values =  msg.split(",");
  if(values.length == 3)
  {
    int x = Integer.valueOf(values[0]);
    int y = Integer.valueOf(values[1]);
    int colorToSet = Integer.valueOf(values[2]);

    this.getController().incomingPixel(x, y, colorToSet);
  }
  else
  {
    int color = Integer.valueOf(msg);
    this.getController().setOwnColor(color);
  }
}
```

Listing 1: parseMessage(String msg)

Erstellen Sie eine Methode `void sendMessage(String message)`, die eine als Parameter übergebene Nachricht an den Server schicken kann.

Aufgabe 4: Login GUI (5P)

Erstellen Sie im Package `de.uks.pmws1213.whiteboard.client.gui` eine grafische Oberfläche mit der man sich an einem Server anmelden kann. Orientieren Sie sich dabei an dem in Abbildung 1 dargestellten Mockup.

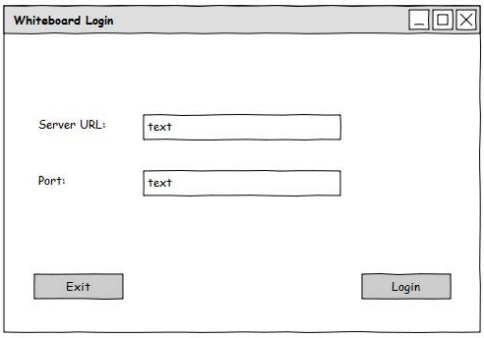

Abbildung 1: Mockup zum Login

Aufgabe 5: Chat GUI (5P)

Erstellen Sie im Package `de.uks.pmws1213.whiteboard.client.gui` eine grafische Oberfläche auf der sich eine Zeichenoberfläche (*SWT Komponete Label*) befindet, über welche eigene Eingaben zum Server gesendet und empfangene Eingaben dargestellt werden. Zusätzlich sollen die Möglichkeiten angeboten werden ein beschriebenes Whiteboard zu leeren oder die Anwendung zu verlassen. Orientieren Sie sich dabei an dem in Abbildung 2 dargestellten Mockup.

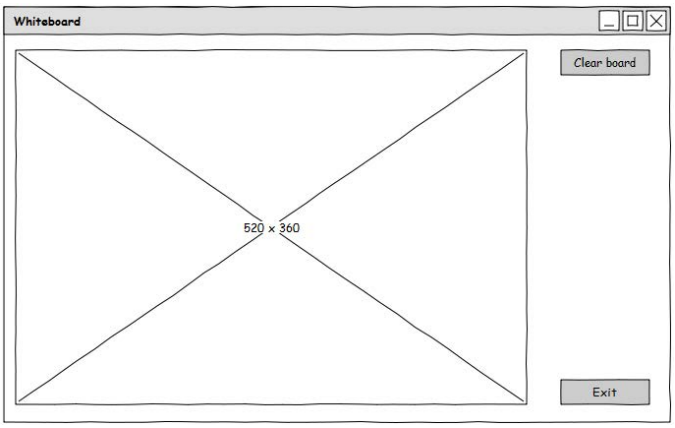

Abbildung 2: Mockup zum Whiteboard

Aufgabe 6: LoginController (5P)

Erstellen Sie im Package `de.uks.pmws1213.whiteboard.client.controller` einen `LoginController`, der zunächst sinnvolle Startwerte für die Felder “Server URL"(z.B. “localhost") und “Port"(z.B. “9000") festlegt. Klickt der Benutzer auf den Login-Button, soll zunächst eine Instanz der in Aufgabe 5 erstellten Whiteboard GUI erzeugt und dann an den `ClientController` übergeben werden. Anschließend kann der `LoginController` gestoppt und der `ClientController` gestartet werden. Klickt der Benutzer auf den Exit-Button, soll die Anwendung beendet werden. Zusätzlich soll eine Klasse `StartClient` im Package `de.uks.pmws1213.whiteboard.client` angelegt werden, welche eine `main`-Methode anbietet, in der ein `LoginController` erstellt und gestartet wird.

Aufgabe 7: ClientController (5P)

Erstellen Sie im Package `de.uks.pmws1213.whiteboard.client.controller` einen `ClientController`, der im Konstruktor die in Aufgabe 5 erstellte Whiteboard GUI, sowie die in die Login GUI eingegebenen Werte für “Server Adresseünd “Port"übergeben bekommt.

Die Klasse benötigt zusätzlich die Attribute `Display display`, `Image image`, `GC grapicContext` und `boolean mousePressed`. Im Konstruktor sind die Attribute folgendermaßen zu belegen:

- `display` bekommt das Display der WhiteboardGUI zugewiesen, welches über die entsprechende get-Methode erreicht werden kann.
- `image` wird initialisiert: `this.image = new Image(display, 520, 360);`
- `graphicContext` wird initialisiert: `this.image = new GC(image);`
- `mousePressed` wird initialisiert: `this.image = false;`

Der `ClientController` ist dafür zuständig die Verbindung zum Server über den in Aufgabe 3 erstellten `ClientNetworkHandler` aufzubauen. Empfängt der `ClientNetworkHandler` eine Nachricht vom Server und leitet sie über die Methode `incomingPixel(..)` an den `ClientController` weiter, muss sie auf ihren Inhalt geprüft werden:

- Ist der Farbwert 0, soll das gesamte Bild weiß gefüllt werden:

```
Color bgColor = new Color(display, 255, 255, 255);
gContext.setBackground(bgColor);
gContext.fillRectangle(x1, y1, x2, y2);
```

- Ist der Farbwert ungleich 0, soll an der Stelle x,y ein kleiner Punkt mit der Farbcodierung “color"gezeichnet werden, diese entspricht der Farbnummer, die dem sendenden Client durch den Server zugewiesen wurde:

```
Color bgColor = new Color(display, 255, 255, 255);
bgColor.handle = color;
gContext.setBackground(bgColor);
gContext.fillArc(x, y, 5, 5, 0, 360);
```

In beiden Fällen muss auf der Zeichenoberfläche das aktuelle `image` gesetzt werden. Dieser Aufruf sollte wie folgt gekapselt werden:

```
display.asyncExec(new Runnable()
{
  public void run()
  {
      // image in das Label setzen
  }
});
```

Listing 2: diyplay.asyncExec(..)

Nun müssen noch Eingaben auf der eigenen Zeichenoberfläche festgestellt und weitergeleitet werden. Dazu benötigt das Label einen `MouseListener` und einen `MouseMoveListener`:

- Der `MouseListener` registriert wenn eine Maustaste gedrückt wird während sich der Mauszeiger über dem Label befindet. Wird eine Maustaste gedrückt, soll eine Klassenvariable `mousePressed` vom Typ boolean auf `true` gesetzt werden. Wird eine Maustaste losgelassen, soll `mousePressed` auf `false` gesetzt werden.
- Der `MouseMoveListener` registriert wenn sich der Mauszeiger über das Label bewegt und wird für jede Positionsänderung einmal aufgerufen. Wenn gleichzeitig die Klassenvariable `mousePressed` auf `true` gesetzt ist, kann davon ausgegangen werden, dass gezeichnet wird. Die aktuellen x- und y-Werte können über `e.x` und `e.y` bezogen werden. Nun soll über den `ClientNetworkHandler` eine Nachricht gesendet werden, die sich aus diesen Teilen zusammensetzt:
  String message = `e.x` + “,"+ `e.y` + “,"+ `ownColor`;

Der Clear-Button soll über den `ClientNetworkHandler` die Nachricht “0,0,0ßenden. Da der Farbwert 0 ist, wird diese von alle Clients als “Zeichenbereich leerenïnterpretiert.

Der Exit-Button ist analog zum Exit-Button aus Aufgabe 6 zu implementieren.