Übungsaufgabe #4: Whiteboard - Client/Server
(Abgabe bis , 27. Mai 2002 16:00) 25 Punkte 06.5.2002

In dieser Aufgabe soll das Whiteboard aus der vorherigen Aufgabe so erweitert werden, dass eine Zeichnung von mehreren Rechnern aus bearbeitet werden kann. Kopieren Sie dazu alle Dateien aus Aufgabe 3 in das Verzeichnis `aufgabe4`. Es kann auch die Musterlösung verwendet werden, die unter `/proj/i4oovs/pub/aufgabe3` (ab 11.05.2002) verfügbar ist. Zur Lösung der Aufgabe werden die aus der Tafelübung bekannten Datenströme (Streams), die Netzwerkkommunikation über Sockets und Threads benötigt.
Um auf die Zeichnung von verschiedenen Rechnern aus zugreifen zu können, soll das Whiteboard in einen Server-Teil und einen Client-Teil aufgeteilt werden. Der Server verwaltet ein Model der Zeichnung, dieses wird durch eine Klasse vom Typ `ShapeContainer` repräsentiert. Die Clients sollen die Darstellung und Steuerung übernehmen. Wird die Zeichnung am Client verändert so wird der Server über diese Änderungen informiert und ändert das Model der Zeichnung. Anschließend informiert der Server alle beteiligten Clients.

Die Aufgabe ist zur einfacheren Bearbeitung in folgenden Teilaufgaben untergliedert:

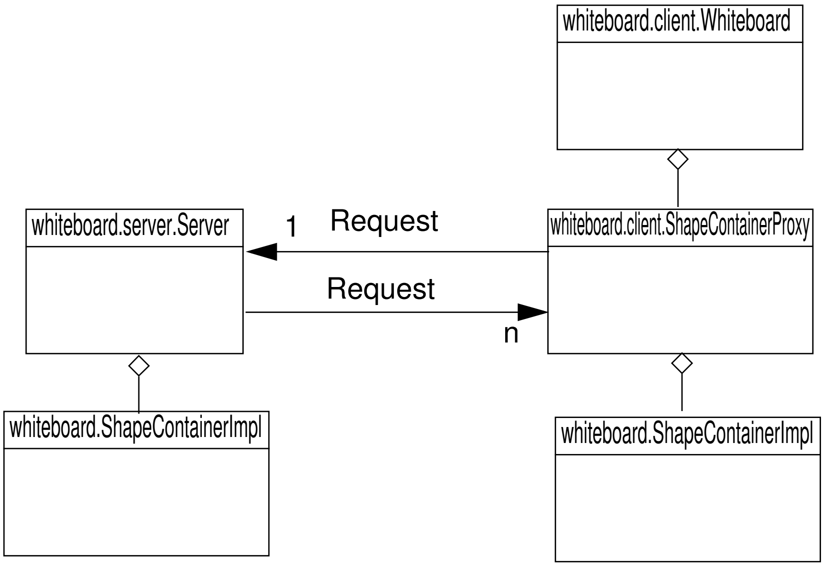
a) Die Klassen sollen für die Aufgabe auf Pakete aufgeteilt werden. Die Aufteilung soll wie folgt geschehen:
   - Das Paket `whiteboard` soll alle Klassen enthalten die sowohl vom Server als auch vom Client genutzt werden.
   - Das Paket `whiteboard.server` soll alle Klassen enthalten, welche nur vom Server genutzt werden.
   - Das Paket `whiteboard.client` enthält die Klassen die nur vom Client genutzt werden.
   - Das Paket `whiteboard.shapes` soll alle Shape-Klassen enthalten.

b) Die Funktionalität der Methode `addShape(String spec)` soll aus dem ShapeContainer entfernt werden und in die Shape-Basisklasse verlagert werden. Hierzu implementieren Sie eine Methode `public static Shape createShape(String spec)` in der Klasse `Shape`. Desweiteren benötigen wir zum eindeutigen Referenzieren von Shape-Objekten eine global eindeutige ID, da Objektreferenzen über verschiedene JVMs hinweg nicht gültig sind. Die Klasse soll dafür eine Klassenvariable `id` besitzen. Implementieren sie zum Vergleichen von Objekten noch die Methode `public boolean isIdentical(Shape s)`, welche an hand der ID zwei identische Objekte erkennt.

c) Passen Sie das Interface `ShapeContainer` und die Klasse `ShapeContainerImpl` wie folgt an. Das Interface `ShapeContainer` soll um folgende Methoden ergänzt werden:
   `public void addShape(Shape s);`
   Diese Methode ersätzt die Methode `addShape(String spec)`.
   `public Vector getShapes();`
   Diese Methode soll einen `Vector` mit allen Shape-Objekten die im Container enthalten sind zurück liefern.
   `public void updateShape(Shape s);`
   Da beim Verändern von Objekten (z.B. Verschieben) nur das lokal vorhandene Objekt verändert wird, nicht jedoch die Objekte auf anderen Rechnern, ist es nötig nach dem Verändern den ShapeContainer über veränderte Objekte zu informieren. Die Methode `updateShape()` soll dieses übernehmen. In Teilaufgabe e) wird diese Methoden benötigt.
   Um beliebig viele Shapes verwalten zu können sollen die Shape-Objekte in der Klasse `ShapeContainerImpl` nun in einem `Vector` anstelle eines Arrays gespeichert werden.

d) Die Kommunikation zwischen den Clients und dem Server soll über Socket-Verbindungen geschehen. Implementieren Sie hierzu eine Klasse `Server` die an einem bestimmten Port Verbindungen entgegen nimmt. Als Portnummer soll Ihre Benutzerkennung aus dem CIP-Pool dienen, diese können Sie mit dem Programm `id` ermitteln. Für jede geöffnete Verbindung soll der Server jeweils einen Thread erzeugen, welcher Anfragen vom Client entgegen nimmt und die Änderungen in das interne Model der Zeichnung übernimmt. Das Model soll durch die Klasse `ShapeContainerImpl` repräsentiert werden.

Über die Verbindung werden Anfragen in der Form von `Request`-Objekten ausgetauscht. Eine Implementierung der `Request`-Basisklasse finden sie unter `/proj/i4oovs/pub/aufgabe4`. Der jeweilige Thread liest vom Socket über einen Objektstream die `Request`-Objekte ein und ruft an diesem die Methode `perform(ShapeContainer)` auf.
Um alle Clients über Veränderungen am Model zu Informieren dient ein weiterer Thread. Dieser wird von allen anderen Threads über eingehende Anfragen informiert und sendet die `Request`-Objekte an alle Clients weiter. Wenn ein neuer Client sich mit dem Server verbindet so bekommt er von diesem Thread alle Shapes mit Hilfe von `AddRequest`-Objekten übermittelt.



e) Verändern Sie die Klasse `WhiteBoard` nun so, dass die Klasse anstelle von `ShapeContainerImpl` eine Klasse `ShapeContainerProxy` benutzt. Diese Klasse implementiert ebenfalls das Interface `ShapeContainer`. Beim Erzeugen einer Instanz soll eine Socket-Verbindung zum Server aufgebaut werden. Über diese Verbindung werden Veränderungen in Form von `Request`-Objekten weiter geleitet.
Der Proxy soll das Model der Zeichnung cachen, um bei einem `paintDrawing()`-Aufruf nicht beim Server anfragen zu müssen. Der `ShapeContainerProxy` besitzt hierfür ein Objekt vom Typ `ShapeContainerImpl` und bearbeitet mit Hilfe eines eigenen Threads ankommende Veränderungsanfragen vom Objektstream.
Die Methoden `updateShape()`, `addShape()` und `deleteShape()` sollen so implementiert werden, dass jeweils eine Anfrage an den Server geschickt wird. Durch den Callback des Servers wird anschließend auch der Cache vom Client verändert und die Aktualisierung der Darstellung am Bildschirm angestossen.
Die Methode `updateShape()` soll nach dem Verschieben eines Objektes aufgerufen werden und die Veränderung in allen Modellen anstossen.

Noch einige Tipps zum Schreiben der Client/Server Anwendung:

- Ein Objektstream überträgt den Zustand eines Objektes nur einmal. Anschließend werden nur noch eine symbolische Referenz übertragen. Um die Zustandsänderung eines Objektes zu übertragen kann entweder jeweils eine Objektkopie (`clone()`) übertragen, oder mit der Methode `reset()` der Objektstrom zurückgesetzt werden.
- Wird von der Benutzerschnittstelle eine Ausnahme ausgelöst, so kann diese nicht an den Server weitergereicht werden. Darum sollten diese Ausnahmen vom Client behandelt werden.
- Swing ist nicht Multithreadfähig.

Die Aufgabe kann nur gewertet werden, wenn sie mit dem abgabe-Programm pünktlich abgegeben wurde. (/proj/i4oods/pub/abgabe aufgabe4)

Die Abgabe ist möglich bis, 27. Mai 2002 16:00 Uhr.