Client-serveur 2020-2021

pfister@connecthive.com

05/10/2020

TP2 – Un premier client et un serveur de figures géométriques

Objectifs : Créer un client en java en utilisant les sockets. Remplacer Putty du TP1 par ce client. Créer un parseur sur le serveur pour décoder des commandes, appeler une méthode métier et retourner le résultat en tant que réponse à la requête.

Prérequis : TP1. Récupérer le corrigé sur le dépôt GIT.

Nota :

Le client doit être utilisé avec le serveur "cs_2019_tp2_circle_server"

Ce dernier ne peut servir qu'un seul client simultanément, ce qui est problématique d'autant plus qu'une connexion est maintenue pour le client qui y met fin avec la commande "quit"

Le serveur peut décoder les commandes :

date

circle x y radius

Il donne des réponses correspondant à ces requêtes en accord avec le document (hypothétique) des spécifications fonctionnelles détaillées issu de l'ingénierie des exigences.

En fonction de la réponse du serveur, le client appelle les opérations nécessaires pour satisfaire aux spécifications.

Des tests de non régression doivent vérifier la conformité des réponses du serveur.

N-B: Le client peut faire tomber le serveur en émettant la commande "stop".

Travail à faire :

- Coder le client de la diapo 35. Le tester avec le serveur LoopServer1 du TP1 sur le même poste et ensuite à partir d'un autre poste du réseau. (attention au numéro de port)
- Que se passe-t-il sur le serveur lorsqu'un client se déconnecte ? Quelle modification pour y remédier ?
- Modifier le client pour qu'il émettre une commande avec argument, exemple [circle x y r].
- Parser cette commande sur le serveur et renvoyer une réponse pertinente en s'aidant des éléments ci-dessous.

```
public class Point {
	int x,y;

	public Point(int x, int y) {
		this.x = x;
		this.y = y;
	}

	public Point rotate(Point point, double theta) {
		double cos = Math.cos(theta);
		double sin = Math.sin(theta);
		int dx = point.x - this.x;
		int dy = point.y - this.y;
		return  new Point(this.x + (int) (dx * cos - dy * sin),this.y + (int)
(dx * sin + dy * cos));
	}

	public double distance(Point other) {
		return Math.sqrt((this.x - other.x) * (this.x - other.x) + (this.y -
other.y) * (this.y - other.y));
	}

	public List<Point> circle(Point point, int step){
		List<Point> result = new ArrayList<Point>();
		for (int th = 0; th < 360; th += step)
			result.add(rotate(point, Math.toRadians(th)));
		return result;

	}
}



	private static String circle(String[] request) {
		String x = request[1];
		String y = request[2];
		String radius = request[3];
		Point center = new Point(Integer.parseInt(x), Integer.parseInt(y));
		Point rad = new Point(Integer.parseInt(x) + Integer.parseInt(radius),
Integer.parseInt(y));
		List<Point> circle = center.circle(rad, 10);
		String response = "";
		for (Point point : circle) {
			response += point.x + ":" + point.y;
			response += ";";
		}
		return response;
	}
```