Πανεπιστήμιο Πελοποννήσου

Σχολή Πληροφορικής και Τηλεπικοινωνιών

Τελική Εργασία

Συγγραφείς:
Αργυρός Κωνσταντίνος
2022202000014

Αναστάσιος Τετράδης
2022202000206

Υπεύθυνος Καθηγητής:
Ν. Πλατής

Η εργασία κατατέθηκε για το μάθημα:
Προηγμένα Θέματα Προγραμματισμού

16 Οκτωβρίου 2023

Περιεχόμενα

Κεφάλαιο 1

Σχετικά με την εργασία

Σε αυτό το κεφάλαιο αποτυπώνονται όλα όσα πρέπει να αναφερθούν πριν την περιγραφή των προγραμμάτων για την καλύτερη κατανόηση του περιεχομένου της αναφοράς.

1.1 Στόχοι Εργασίας

1.1.1 Βασικοί Στόχοι Εργασίας

Βασικός στόχος της εργασίας είναι η πλήρη ανάπτυξη και υλοποίηση ενός ζεύγους προγραμμάτων για μία επιχείρηση πλυντηρίου οχημάτων. Το πρώτο από τα δύο αυτά προγράμματα, ο λεγόμενος Client, είναι ένα πρόγραμμα που θα τρέχει σε μια οθόνη προσβάσιμη από χρήστες (πελάτες του μαγαζιού). Αυτός θα καταγράφει τις επιλογές των χρηστών, θα τις καταχωρεί σε παραγγελίες και θα τις στέλνει στο δεύτερο πρόγραμμα, τον Server. Ο Server, είναι ένα πρόγραμμα που θα τρέχει στον υπολογιστή του ταμείου της επιχείρησης. Αυτός θα λαμβάνει τις παραγγελίες από τον Client και, με βάση την είσοδο του ταμία, θα τις καταχωρεί ή θα τις διαγράφει από ένα αρχείο εσόδων.

1.1.2 Περαιτέρω Στόχοι Εργασίας

Για την ευρύτερη χρήση των προγραμμάτων, έγινε προσπάθεια τα προγράμματα (κυρίως του Client) να προσφέρουν ένα κομμάτι της προσαρμοστικότητάς τους στον πελάτη[1] χωρίς την ανάγκη τροποποίησης του πηγαίου τους κώδικα. Έτσι, ο πελάτης μπορεί να τροποποιήσει ορισμένα στοιχεία της διεπαφής και των δεδομένων προς την προτίμησή του χωρίς να χρειάζεται να ξέρει τίποτα σχετικά με τις τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση των προγραμμάτων (Java/JavaFX/FXML). Ως συνέπεια αυτού όμως, η διεπαφή πρέπει να παράγεται με δυναμικό τρόπο έτσι ώστε να υποστηρίζονται όλες οι αλλαγές της διεπαφής που θέλει να υλοποιήσει ο πελάτης. Έτσι, για παράδειγμα, ο χρήστης μπορεί να προσθέσει τους δικούς του τύπους οχημάτων, έξτρα υπηρεσίες, τα δικα του εικονίδια ή ακόμα και να αλλάξει τα ήδη υπάρχων δεδομένα (πχ. τιμή, όνομα υπηρεσίας) ή και τη σειρά με την οποία εμφανίζονται.

1

1.2 Σημαντικές Ορολογίες

Για την αποφυγή σύγχυσης, αναφέρονται και εξηγούνται ορισμένες έννοιες οι οποίες είναι σημαντικό να ξεκαθαριστούν από νωρίς.

- **Ομάδα ανάπτυξης**, των προγραμμάτων αποτελείται από τους συγγραφείς της αναφοράς.
- **Πελάτης**, είναι το άτομο που αποκτά το ζέυγος προγραμμάτων από την ομάδα ανάπτυξης (πχ. ο διευθυντής της επιχείρησης.)
- **Χρήστης**, είναι το άτομο που θα χρήσιμοποιήσει ένα (τουλάχιστον) από τα δύο προγράμματα (πχ. πελάτης της επιχείρησης για τον Client, ταμίας για τον Server).
- **Παραγγελία (order)**, είναι το σύνολο των υπηρεσιών που επιλέγει και ζητά ένας χρήστης, συνοδευμένο από περαιτέρω δεδομένα (πχ. αριθμό κυκλοφορίας, τύπος οχήματος κλπ.)
- **Υπηρεσία (service)**, είναι μια ενέργεια συντήρησης/εκκαθάρισης για συγκεκριμένο όμως τύπο και περιοχή/εξάρτημα ενός οχήματος (πχ. εξωτερικό καθάρισμα αυτοκινήτου **δεν** είναι η ίδια υπηρεσία με το εξωτερικό καθαρισμό για 4X4 Jeep)
- **Ομάδα Υπηρεσιών (service group)**, είναι μια ομαδοποίηση ενός συνόλου παρόμοιων υπηρεσιών (πχ. το "εξωτερικό καράρισμα" περιλαμβάνει απλή, ειδική και βιολογική έκδοση, αλλά και οι τρείς εκδοχές αποτελούν τρείς διαφορετικές υπηρεσίες).

1.3 Δομή Εργασίας

Για την καλύτερη κατανόηση της περιγραφής της λειτουργικότητας των προγραμμάτων, θα επεξηγηθεί η γενική δομή της εργασίας, καθώς και ο ρόλος της κάθε κλάσης και αρχείου.

1.3.1 Γενική Δομή Εργασίας

Η εργασία χωρίζεται σε 2 επιμερούς φακέλους, ένα για τον Client και ένα για τον Server. Και για τους δύο φακέλους, οι σημαντικότεροι υποφάκελοι είναι

1. ο φάκελος "java" που περιέχει όλες τις προσαρμοσμένες (custom) κλάσεις που χρησιμοποιούνται από το πρόγραμμα και
2. ο φάκελος "resources" που περιέχει όλο το υλικό που χρησιμοποιεί το πρόγραμμα, όπως αρχεία διεπαφής (fxml), στυλιστικά αρχεία (css), αρχεία εικόνων/εικονιδίων (png) και αρχεία δεδομένων χρήστη (προσαρμοσμένα xml και png).

1.3.2 Περιγραφή Κλάσεων

Οι κλάσεις που δημιουργήθηκαν για την ορθή λειτουργία των προγραμμάτων είναι οι εξής:

- **Service**, είναι η κλάση που αναπαριστά μία υπηρεσία. Κάθε υπηρεσία έχει ένα όνομα, μία τιμή και μια αναφορά στην ομάδα (group) που ανήκει.
- **ServiceGroup**, είναι η κλάση που αναπαριστά μια ομάδα από υπηρεσίες. Κάθε ομάδα (group) έχει ένα όνομα, ένα αναγνωριστικό εικονιδίου και μια λίστα από υπηρεσίες
- **Vehicle**, είναι η κλάση που αναπαριστά ένα τύπο οχήματος που υποστηρίζει η επιχείρηση. Κάθε τύπος οχήματος έχει ένα όνομα, ένα αναγνωριστικό εικονιδίου και μια λίστα από ομάδες διαθέσιμων υπηρεσιών.

- **ServiceDB**, είναι η κλάση που αναπαριστά μια δομή που κρατά όλες τις πληροφορίες για όλες τις διαθέσιμες υπηρεσίες που προσφέρει η επιχείρηση. Περιέχει ένα σύνολο από τύπους οχημάτων.[2] Δεδομένου ότι η κλάση αυτή χρειάζεται να κρατάει μια συλλογή από τύπους οχημάτων **χωρίς αντίγραφα**, μια δομή τύπου Set είναι κατάλληλη. Ο λόγος όμως χρήσης του **LinkedHashSet** συγκεκριμένα είναι επειδή η δομή αυτή διατηρεί και την σειρά με την οποία εισάγονται τα δεδομένα. Αυτό καθιστά δυνατό έμμεσα στον πελάτη να έχει έλεγχο όχι μόνο στα δεδομένα που θέλει να προβάλλει, αλλά και στη **σειρά με την οποία τα δεδομένα αυτά εμφανίζονται**.

- **Order**, είναι η κλάση που αναπαριστά μια παραγγελία. Κάθε παραγγελία έχει τον αριθμό κυκλοφορίας του χρήστη, το όνομα του ζητούμενου τύπου οχήματος, μια λίστα από ζητούμενες υπηρεσίες, το συνολικό κόστος των υπηρεσιών αυτών και μια ημερομηνία άφιξης και ολοκλήρωσης.

- **OrderQueue**, είναι μια κλάση που χρησιμοποιείται μόνο από τον Server και αναπαριστά μια συλλογή από παραγγελίες. Έχει ως πεδίο μια παρατηρήσιμη (observable) λίστα από παραγγελίες.

- **Controller** κλάσεις, είναι όλες οι κλάσεις που το όνομα τους τελειώνει σε "Controller" (πχ. RegistrationNumberController). Κάθε Controller κλάση είναι υπεύθυνη για την συμπεριφορά **μίας** σελίδας.

- **Client**, είναι η βασική κλάση του Client προγράμματος και περιέχει όλη την κύρια λειτουργικότητά του. Η κλάση αυτή είναι επίσης υπεύθυνη και για την αλλαγή οθονών.

- **Server**, αντίστοιχα με την κλάση Client παραπάνω, είναι η βασική κλάση του Server προγράμματος και περιέχει όλη την κύρια λειτουργικότητά του.

[2]Περιέχει και αναγνωριστικά χρωμάτων, το οποία όμως χρησιμοποιούνται μόνο για debugging.

1.3.3 Περιγραφή Αρχείων

Τα αρχεία που χρησιμοποιούν τα προγράμματα είναι τα εξής:

- Αρχεία διεπαφής **.fxml** (resources\gr\uop), είναι τα αρχεία με οποία έχουν υλοποιηθεί όλες οι οθόνες των προγραμμάτων.
- Στυλιστικά Αρχεία **.css** (resources\gr\uop\stylesheets), είναι τα αρχεία με τα οποία έχουν ορίζουν εμφανισιακά ορισμένα στοιχεία των οθονών.
- Αρχεία γραμματοσειρών**.ttf** (resources\gr\uop\stylesheets\fonts) είναι τα αρχεία τα οποία προσφέρουν προσαρμοσμένες γραμματοσειρές στην διεπαφή.
- Αρχεία εικονιδίων **.png** (resources\gr\uop\data) είναι τα αρχεία με τα οποία φορτώνονται τα εικονίδια *που θέλει ο πελάτης* στην διεπαφή.
- Αρχεία διαμόρφωσης **config.xml** (resources\gr\uop\data) είναι τα αρχεία με τα οποία ο πελάτης έχει την δυνατότητα να προσαρμόσει ορισμένα στοιχεία και δεδομένα του προγράμματος.

1.4 Αναφορές Πηγών

Κατά την διαδικασία υλοποίσης της εργασίας, χρησιμοποιήθηκαν εξωτερικές πηγές για την υλοποίηση ορισμένων αντικειμένων (πχ. για τα Threads). Για την αποφυγή αντιγραφής και λογοκλοπής, οι περισσότερες (αν όχι όλες) οι πηγές έχουν συμπεριληφθεί ως σχόλια στον πηγαίο κώδικα, στα αντίστοιχα σημεία που αναφέρονται. Οι σύνδεσμοι αυτοί επίσης συμπεριλαμβάνονται και στο τέλος της αναφοράς ως "πηγές" (references).

Κεφάλαιο 2

Λειτουργικότητα Προγραμμάτων

Σε αυτό το κεφάλαιο θα αναλυθεί ο τρόπος λειτουργίας των Client και Server.

2.1 Λειτουργικότητα Client

2.1.1 Γενική Περιγραφή Δομής

Ο Client περιέχει (ως static πεδία) μια δομή με όλες τις πληροφορίες των διαθέσιμων υπηρεσιών από τον πελάτη[1] (ServiceDB), μια δομή για την αποθήκευση των επιλογών του χρήστη (Order), δομές αντιστοιχίας κουμπιών - δεδομένων[2] και δομες παρατήρησης αλλαγής επιλογών του χρήστη.

[1] Διευθυντή πλυντηρίου.

[2] Η χρησιμότητά τους περιγράφεται αναλυτικότερα παρακάτω.

2.1.2 Αρχικοποίηση Βάσης Δεδομένων

Το πρόγραμμα του Client ξεκινά διαβάζοντας το αρχείο config.xml. Ο τρόπος ανάνγωσης του αρχείου είναι μια απλή προσπέλαση όλων των XML Tags, όπου το κάθε tag αντιστοιχεί σε κάποιο αντικείμενο του Client (πχ. με το tag <vehicle> φτιάχνεται ένα αντικείμενο τύπου Vehicle). Tag προς tag η δομή δεδομένων συμπληρώνεται ώσπου να τελειώσει το αρχείο. Αν διαπιστωθεί λάθος σύνταξη (formatting) τότε το πρόγραμμα δεν εκκινεί. Επιλέχθηκε η XML σύνταξη (και όχι JSON) επειδή είναι πιο απλή στην κατανόηση από ένα προγραμματιστικά ανέμπειρο άτομο.

2.1.3 Υλοποίηση Φόρτωσης Διεπαφής

Κάθε οθόνη έχει υλοποιηθεί μέσω *τουλάχιστον ενός* FXML αρχείου. Κατα τη διάρκεια φόρτωσης μιας σελίδας, με βάση τα δεδομένα από τη δομή δεδομένων (ServiceDB), καθώς και τις επιλογές του χρήστη (Order), φορτώνονται τα αντίστοιχα (και σε δεδομένα, και σε αριθμό) επιμερούς FXML στοιχεία. Η αλλαγή από σκηνή σε σκηνή γίνεται μέσω του Client.

Τα επιμερούς FXML στοιχεία αντιστοιχούνται με την ανάλογη υπηρεσία τους σε μια δομή αντιστοιχίας. Αυτό γίνεται επειδή σε περίπτωση ενέργειας (πχ. πάτημα κουμπιού) το επιμερούς FXML στοιχείο δε μπορεί να ξέρει άμεσα από τον FXML γονέα του ποιά υπηρεσία αντιστοιχεί σε αυτό (πχ. αν πατηθεί κουμπί επιλογής υπηρεσίας, ποιά υπηρεσία πρέπει να εισαχθεί στην παραγγελία;). Μέσω των δομών αντιστοιχίας, όχι μόνο τα επιμερούς FXML στοχεία λειτουργούν ορθά, αλλά έχουν και την δυνατότητα απομνημόνευσης επιλογών σε περίπτωση οπισθοδρόμησης (για παράδειγμα, αν ο χρήστης δει την τελική παραγγελία και διαπιστώσει ότι ξέχασε να επιλέξει μια υπηρεσία, γυρνώντας πίσω στην προηγούμενη σελίδα, τα στοιχεία που είχε ήδη επιλέξει εμφανίζονται ήδη επιλεγμένα).

2.1.4 Συλλογή Δεδομένων Παραγγελίας

Οι επιλογές του χρήστη δεν αποθηκεύονται απευθείας στην δομή της παραγγελίας (Order). Πρώτα αποθηκεύονται σε μια "παρατηρήσιμη" δομή (Observable) και μετά αποθηκεύονται στην κύρια δομή της παραγγελίας. Αυτό γίνεται επειδή για να μπορέσουν τα επιμερούς FXML στοιχεία μιας οθόνης να "ακούν" για αλλαγές κάποιου στοιχείου/πεδίου (πχ. αριθμός στοιχείων TableView), το πεδίο./στοιχείο αυτό πρέπει αναγκαστικά να είναι static. Όμως, τα FXML πεδία εξ' ορισμού *δεν μπορούν να είναι static*. Επομένως, αποθηκεύοντας τις αλλαγές πρώτα σε μια δομή παρατήρησης και μετέπειτα στη δομή παραγγελίας, δίνεται η δυνατότητα σε όλα τα FXML στοιχεία να αντιδρούν ορθά στις αλλαγές με βάση τις επιλογές του χρήστη.

2.1.5 Αποστολή Παραγγελίας

Κατά την ολοκλήρωση της παραγγελίας, εμφανίζεται η οθόνη επανακεφαλαίωσης, στην οποία ο χρήστης έχει την δυνατότητα να επιθεωρίσει τις επιλογές του, να αφαιρέσει όποια δεν επιθυμεί και τέλος να καταχωρίσει την παραγγελία στον Server. Αν η σύνδεση με τον Server και η αποστολή της παραγγελίας είναι επιτυχή, τότε εμφανίζεται η οθόνη επιβεβαίωσης. Σε διαφορετική περίπτωση, εμφανίζεται η οθόνη σφάλματος, που παρακινεί τον χρήστη να δοκιμάσει να ξαναστείλει την παραγγελία αργότερα.

Η παραγγελία στέλνεται **ως ένα αντικείμενο** στον Server για λόγους απλότητας. Για την αποφυγή λάθους κατά την αποστολή, αποφεύχθηκε η χρήση των δομών παρακολούθησης αλλαγών (Observable) εντός της δομής Order. Για αυτό το λόγο, οι δομές αυτές χρησιμοποιούνται αναγκαστικά στον Client.

2.2 Λειτουργικότητα Server

2.2.1 Γενική Περιγραφή Δομής

Η δομή του Server είναι πολύ παρόμοια με αυτή του Client. Η δομή παραγγελίας (Order), η δομή πληροφοριών (ServiceDB) και όλες οι επιμερούς δομές (Vehicle, ServiceGroup και Service) είναι κοινές με αυτές του Client.

2.2.2 Αρχικοποίηση Βάσης Δεδομένων

Με τον τρόπο που δουλεύει ο Server, τα δεδομένα δεν χρειάζονται για την ορθή λειτουργία του, καθώς κάθε παραγγελίας ήδη κουβαλάει ως πεδίο και το συνολικό της κόστος. Για λόγους όμως ασφάλειας διατήρησης της ακεραιότητας των δεδομένων, προστέθηκε η δομή αυτή μαζί με έναν μηχανισμό αποφυγής απάτης (scam prevention mechanism) ο οποίος περιγράφεται παρακάτω. Κατά την εκκίνηση του Server, η δομή πληροφοριών αρχικοποιείται κανονικά με το config.xml αρχείο, ακριβώς με τον ίδιο τρόπο όπως και στον Client.

2.2.3 Συλλογή Παραγγελιών

Αμέσως μετά την αρχικοποίηση της δομής πληροφοριών, φτιάχνεται ένα Thread το οποίο κοιτάζει επαναληπτικά την θύρα (port) 9999. Με το που σταλθεί παραγγελία από τον client, το thread αυτό το λαμβάνει ως απλό Java αντικείμενο (object), το μετατρέπει (downcast) σε Order, διεσφαλίζει τα δεδομένα του και αν η παραγγελία είναι έγκυρη τότε την προσθέτει στο αρχείο εσόδων και στην δομή ουράς παραγγελιών.

2.2.4 Μηχανισμός Επιβεβαίωσης Ακεραιότητας Παραγγελιών

Η διασφάλιση των δεδομένων της παραγγελίας είναι μια απλή "επανεγγραφή" των τιμών της παραγγελίας. Στη περίπτωση που, με κάποιο τρόπο, ένας χρήστης καταφέρει να παραβιάσει τα δεδομένα της εφαρμογής του Client και να τροποποιήσει τις τιμές των υπηρεσιών προς όφελός του, ο server απλά θα ξαναγράψει (overwrite) τις τιμές των υπηρεσιών της παραγγελίας σύμφωνα με αυτές που θα έπρεπε να είναι. Έτσι αποφεύγονται απάντες και στην περίπτωση που με τον ίδιο τρόπο εισαχθεί στη παραγγελία μη διαθέσιμη υπηρεσία σκοπίμως, η παραγγελία απλά απορρίπτεται.[3]

[3]Σε περίπτωση απόρριψης παραγγελίας, ο χρήστης που έκανε την παραγγελία **δεν** ενημερώνεται.

2.2.5 Καταγραφή Παραγγελιών

Όπως αναφέρθηκε και παραπάνω, οι παραγγελίες που εγγράφονται στο αρχείο εσόδων προσθέτονται επιτόπου στη δομή ουράς παραγγελιών. Παρά το όνομα της, η ουρά παραγγελιών ως δομή είναι μια (παρατηρήσιμη) λίστα από παραγγελίες. Ο λόγος που δεν υλοποιήθηκε ως ουρά Queue είναι επειδή θεωρήθηκε πολύ πιθανή η περίπτωση ολοκλήρωσης μιας παραγγελίας, που εισήλθε τελευταία, πρώτη από άλλες που εισήλθαν πριν από αυτήν. Σε μια τέτοια περίπτωση, θεωρείται δύσκολη την ολοκλήρωση μιας παραγγελίας που βρίσκεται στο τέλος της ουράς. Επίσης, με μια παρατηρήσιμη (Observable) δομή, τα στοιχεία που πρέπει να αντιδρουν στις αλλαγές της ουράς παραγγελιών παρατηρούν τις αλλαγές αυτές πολύ πιο εύκολα.

2.2.6 Υλοποίηση Διεπαφής

Η διεπαφή του Server είναι υλοποιημένη με τον ακριβώς ίδιο τρόπο όπως και με τον Client. Επειδή όμως ο Server, σε αντίθεση με τον Client, δεν έχει διαδραστικά επιμερούς FXML στοιχεία, δεν χρειάζονται (και συνεπώς δεν περιέχει ως πεδία) δομές αντιστοίχης. Ορισμένα στοιχεία αντιδρούν σε αλλαγές της δομής παραγγελιών (πχ TableView) ακούγοντας τη λίστα της ίδιας της δομής απευθείας, χωρίς την χρήση εξωτερικών παρατηρήσιμων δομών όπως στον Client.

Κεφάλαιο 3

Πηγές

3.1 Γραφικά Διεπαφών

1. Figma Application Concept Project https://www.figma.com/file/GrHIr6VidoBMWSuZBkDUW2/Untitled?type=design&node-id=0%3A1&mode=design&t=r1xZMeqV9e17BGBo-1

3.2 Πηγές Εικονιδίων - Εικονογραφήσεων

1. Εικονογραφήσεις https://storyset.com/
2. Εικονίδια https://www.flaticon.com/
3. Εικονίδια https://iconscout.com/

3.3 Προγραμματιστικές Πηγές

1. TableView Information https://docs.oracle.com/javafx/2/ui_controls/table-view.htm
2. LocalDateTime Information #1 https://www.w3schools.com/java/java_date.asp

3. LocalDateTime Information #2 https://www.geeksforgeeks.org/java-time-localdatetime-class-in-java/

4. LinkedHashSet vs LinkedHashMap Difference in Memory. https://stackoverflow.com/questions/65702840/linkedhashmap-vs-linkedhashset-for-retrieving-specific-elements-retrieving-noredirect=1&lq=1

5. Collectors.toSet (NOT USED)https://www.geeksforgeeks.org/collectors-toset-in-java-with-examples/

6. Controller Access Ideas https://stackoverflow.com/questions/10751271/accessing-fxml-controller-class

7. FXMLLoader Handling https://bugs.openjdk.org/browse/JDK-8125877

8. Changing Image in ImageView (Never replace the actual imageView #1) https://stackoverflow.com/questions/29500761/javafx-change-the-image-in-an-imageview

9. Initialize Method (Never replace the actual imageView #2) https://stackoverflow.com/questions/60323494/javafx-if-node-equals-node

10. Contructor vs Initialize https://stackoverflow.com/questions/34785417/javafx-fxml-controller-constructor-vs-initialize-method

11. Using Java to reset an element's style https://stackoverflow.com/questions/30759310/how-to-reset-back-to-default-css-after-adding-style

12. Using Initialize method to set up FXML listeners https://stackoverflow.com/questions/53613385/fxmlloader-use-adding-data-to-a-component-controlled-by-another-controller

13. How to listen to ObservableList changes #1 https://stackoverflow.com/questions/51081000/javafx-listen-to-changes-in-list

14. How to listen to ObservableList changes #2 https://stackoverflow.com/questions/19890250/instantiating-observableintegervalue-in-javafx

15. How to listen to ObservableList changes #3 https://genuinecoder.com/javafx-observable-list-tutorial/

16. Lambda writing for ListChangeListeners https://itecnote.com/tecnote/java-how-to-write-a-new-listchangelistener-with-lambda/

17. NotSerializableException Information https://www.geeksforgeeks.org/notserializableexception-in-java-with-examples/

18. InvalidClassException Information https://stackoverflow.com/questions/10378855/java-io-invalidclassexception-local-class-incompatible

19. How to populate FXML Injected TableView with existing data https://www.tutorialspoint.com/how-to-add-data-to-a-tableview-in-javafx

20. The idea of using observable lists https://stackoverflow.com/questions/41508932/javafx-add-rows-at-the-beginning-of-the-tableview

21. ClassCastException Information (FOR CSS PROBLEM) https://stackoverflow.com/questions/27311222/javafx-getting-class-cast-exception-in-css-for-blend-mode

22. How to properly stop a thread in Java https://stackoverflow.com/questions/10961714/how-to-properly-stop-the-thread-in-java

23. Thread.stop() vs Thread.interrupt() https://stackoverflow.com/questions/5244312/why-doesnt-thread-stop-work-in-situations-where-thread-interrupt-doesnt-wor
rq=3

24. BindException Information https://stackoverflow.com/questions/12737293/how-do-i-resolve-the-java-net-bindexception-address-already-in-use-jvm-binc

25. How to delete a thread in Java #1 https://www.geeksforgeeks.org/killing-threads-in-java/

26. How to delete a thread in Java #2 https://stackoverflow.com/questions/30436082/how-to-delete-a-thread-in-java

27. Thread Still running after program Termination `https://stackoverflow.com/questions/48581777/threads-still-running-after-system-exit`