



Πανεπιστήμιο Κρήτης, Τμήμα Επιστήμης Υπολογιστών

HY252 – Αντικειμενοστρεφής Προγραμματισμός

Εξάμηνο: Χειμερινό 2017-2018

Διδάσκων: Γιάννης Τζίτζικας

Βοηθοί: Γ. Νικητάκης, Π. Λιονάκης, Μ. Παπαδάκη

2^η Σειρά Ασκήσεων

Ανάθεση: Τετάρτη 25 Οκτωβρίου 2017

Παράδοση: Τετάρτη 15 Νοεμβρίου 2017

(δείτε τις ημερομηνίες για bonus που περιγράφονται παρακάτω)

Εκπαιδευτικοί στόχοι

Αυτή η σειρά ασκήσεων είναι η πιο σημαντική του μαθήματος.

- Η 1^η άσκηση (**μέσα αποθήκευσης και νέφη**, 20 μονάδες) θα σας επιτρέψει να εξοικειωθείτε με τον ορισμό κλάσεων και υποκλάσεων, τη δημιουργία αντικειμένων (στιγμιότυπων των κλάσεων που ορίσατε), την υποσκέλιση μεθόδων (overriding),
- Η 2^η άσκηση (**τραμπάλα**, 20 μονάδες) θα σας επιτρέψει να εξοικειωθείτε με την προδιαγραφή Αφαιρετικών Τύπων Δεδομένων (Abstract Data Types), με φυσική γλώσσα, και την υλοποίησή τους με κλάσεις Java (και χρήση εξαιρέσεων).
- Η 3^η άσκηση (**Game of Thrones**, 40 μονάδες) θα σας επιτρέψει να εξοικειωθείτε με τις διεπαφές (interfaces), τις αφηρημένες (abstract) κλάσεις και τα συνεργαζόμενα αντικείμενα (Collaborating Objects) - και να κερδίσετε .. αναμνηστικό έπαινο ☺
- Η 4^η άσκηση (**Vending Machine**, 25 μονάδες) θα σας επιτρέψει να εξοικειωθείτε με την προδιαγραφή Αφαιρετικών Τύπων Δεδομένων (με Java interfaces), την τεκμηρίωσή τους με JavaDoc, την υλοποίησή τους με κλάσεις Java, τον έλεγχο ορθότητας της υλοποίησής τους με JUnitTest και θα έρθετε σε επαφή με τα γραφικά της Java

Σημειώσεις

1. **Απορίες** σχετικά με την Α2 θα απαντώνται μόνο μέσω του forum της Α2 στο moodle. Επίσης μην ξεχνάτε ότι κάθε Παρασκευή υπάρχουν ώρες γραφείου.
2. Θα γίνει και **φροντιστήριο** σχετικό με JavaDoc και JUnit. Τα generics θα διδαχθούν αναλυτικά αργότερα, στην παρούσα φάση απλά θα τα χρησιμοποιήσετε.
3. **Ημερομηνία παράδοσης και bonus**: Όσοι παραδώσουν τις **2 πρώτες στις 5 Νοεμβρίου** θα έχουν **10% bonus** στο βαθμό που πήραν. Όσοι παραδώσουν τις **2 τελευταίες ασκήσεις 2 μέρες πριν την προθεσμία** θα έχουν **bonus 10%** σε αυτές τις ασκήσεις.
4. **Οδηγίες Παράδοσης**. Για λεπτομέρειες κοιτάξτε στην επόμενη σελίδα. **ΠΡΟΣΟΧΗ:** (α) Πρέπει να παραδώσετε τα **.java αρχεία, δηλαδή τα αρχεία που περιέχουν τον κώδικά σας, και όχι τα εκτελέσιμα .class αρχεία!** (β) **Ακολουθήστε επακριβώς τις οδηγίες παράδοσης που αναγράφονται στο τέλος της εκφώνησης.** Διαφορετικά η διόρθωση είναι πολύ χρονοβόρα για τους μεταπτυχιακούς οι οποίοι έχουν και αυτοί μαθήματα και εργασία να κάνουν. **Αν παραδώσετε κάτι που δεν είναι σύμφωνο με τους κανόνες παράδοσης τότε θα έχετε 40% μείωση του βαθμού και ίσως η άσκησή σας δε θα βαθμολογηθεί.**
5. Η **εξέταση** της Α2 θα γίνει μέσω μίας υποχρεωτικής εργαστηριακής άσκησης (**A2E**). Θα είναι μια απλή και μικρή άσκηση που θα πρέπει να κάνετε στο εργαστήριο/αναγνωστήριο υπό την εποπτεία των βοηθών του μαθήματος (σε περίπτωση απουσίας ή αποτυχίας δεν θα προσμετρηθούν οι βαθμοί της Α2 – για ειδικές περιπτώσεις (π.χ. απουσίες λόγω θεμάτων υγείας) υπάρχει σχετικό forum).

Προτεινόμενος Χρονοπρογραμματισμός

Χρονικό Διάστημα	Άσκηση
25 Οκτ – 5 Νοεμ. (bonus)	1, 2
5 – 15 Νοεμ. (13 Νοεμ. για bonus)	3, 4
15 Νοεμ.- 24 Νοέμ.	Μελέτη για πρόοδο



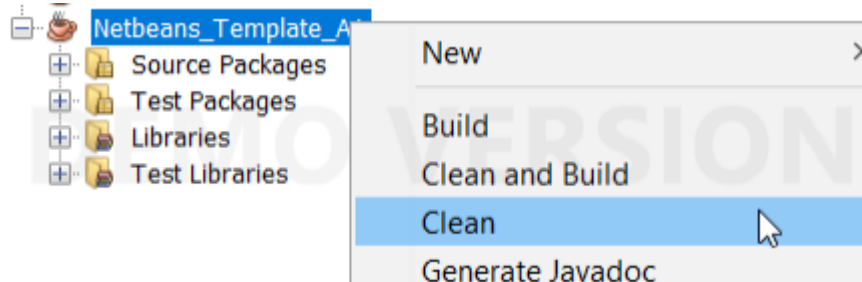
ΟΔΗΓΙΕΣ ΟΡΓΑΝΩΣΗΣ ΤΩΝ ΠΑΡΑΔΟΤΕΩΝ ΑΡΧΕΙΩΝ

Βήμα 1

Ανάλογα με το IDE που έχετε, χρησιμοποιήστε ένα από τα τρία project templates που σας παρέχουμε (Netbeans_Template_A2, Eclipse_Template_A2 και IntelliJ_Template_A2). Τα templates αυτά περιέχουν όλα τα απαραίτητα πακέτα (packages) καθώς και τις κλάσεις (classes) που καλείστε να υλοποιήσετε. **ΜΗΝ αλλάξετε τη δομή αυτή**, απλά συμπληρώστε καταλλήλως τις κλάσεις με τις μεθόδους που σας ζητούνται καθώς και με δικές σας σε περιπτώσεις που επιθυμείτε να κάνετε καλύτερη διαχείριση του κώδικα σας.

Βήμα 2

Προτού παραδώσετε κάνετε clean το project.



Βήμα 3

Το όνομα του project σας θα πρέπει να ακολουθεί την εξής μορφή:

<IDE_name>_A2_<Αριθμός_Μητρώου>. Π.χ. στην περίπτωση που χρησιμοποιείτε Netbeans θα είχαμε Netbeans_A2_1234

Βήμα 4

Συμπίεστε το project σας και παραδώστε το μέσω του moodle.



Άσκηση 1. Μέσα Αποθήκευσης και Νέφη

Αξία: **20 μονάδες**

Συνδεδεμένες κλάσεις, δημιουργία αντικειμένων, overriding

Προσοχή (για τα παραδοτέα): Ακολουθήστε πιστά τη δομή του project που σας έχει δοθεί και συμπληρώστε καταλλήλως τα αρχεία βάσει της εκφώνησης που ακολουθεί.

(α) [10 μονάδες]

Ορίστε μια κλάση **StorageMedium** με δύο ιδιωτικά πεδία: **name** (String) και **capacity** (int) και **type** θεωρώντας μια δήλωση της μορφής `enum Type {CD, DVD, FlashDrive, HD, SSD}` η οποία να έχει ως αρχική τιμή την τιμή `Type.HD`. Συμπληρώστε τις μεθόδους που απαιτούνται βάσει της αρχής της ενθυλάκωσης και ορίστε κατάλληλα μια κατασκευάστρια μέθοδο που να έχει δύο παραμέτρους (name και capacity). Ορίσετε μία δημόσια μέθοδο `String toString()` η οποία θα επιστρέφει ένα string με τις τιμές των name και capacity με κατάλληλη, για ανάγνωση, μορφή, π.χ.



«Αποθηκευτικό μέσο: <όνομα αντικειμένου>, χωρητικότητα: <capacity αντικειμένου> bytes, Τύπος <type>.»

Ορίστε μια υποκλάση της `StorageMedium` με όνομα **USBstick** της οποίας η `String toString()` να επιστρέφει τη συμβολοσειρά "USB Stick " ακολουθούμενη από αυτό που επιστρέφει η κλήση της `toString()` της υπερκλάσης `StorageMedium`. Κάντε ό,τι άλλο είναι απαραίτητο για να μεταγλωττίζεται ο κώδικας. Η τιμή του `type` να είναι αρχικοποιημένη (στην κατασκευή του αντικειμένου) στην τιμή `FlashDrive`.

(β) [10 μονάδες]

Δημιουργείστε μια κλάση **CloudStorage** η οποία θα έχει έναν πίνακα με όνομα **components** με 100 θέσεις που να μπορούν να δείχνουν σε στιγμιότυπα της κλάσης `StorageMedium`. Η κλάση `CloudStorage` να έχει μια μέθοδο **init** η οποία θα γεμίζει τον πίνακα `components` ώστε τα πρώτα 50 κελιά του να δείχνουν σε 50 νέα στιγμιότυπα της κλάσης `USBstick`, και τα υπόλοιπα 50 κελιά του να δείχνουν σε 50 νέα στιγμιότυπα της κλάσης `StorageMedium`. Τα ονόματα (`name`) των (άμεσων ή έμμεσων) στιγμιότυπων της `StorageMedium` που θα δημιουργηθούν πρέπει να είναι διαφορετικά, συγκεκριμένα από το `SM1` (για το 1ο) έως `SM100` (για το τελευταίο). Για την τιμή του γνωρίσματος `capacity` να είναι ένας τυχαίος αριθμός από το 1000 έως 100.000. Στο τέλος η `init` πρέπει να διατρέχει τον πίνακα `components` και να καλεί την `toString` του κάθε αντικειμένου (εκτυπώνοντας την επιστρεφόμενη τιμή στην κονσόλα) που δημιουργήθηκε.

Δημιουργήστε συνάμα μια δημόσια μέθοδο `getCapacity()` η οποία να επιστρέφει την συνολική ικανότητα αποθήκευσης (αθροίζοντας το `capacity` των αντικειμένων στον πίνακα `components`).

Τέλος φτιάξτε μια κλάση **Tester** της οποίας η `main` να δημιουργεί ένα στιγμιότυπο της `CloudStorage` και αν συνεχεία να καλεί τη μέθοδο `init` αυτού του αντικειμένου. Απόσπασμα εκτέλεσης της `main` της `Tester` ακολουθεί:

```
USB stick Αποθηκευτικό μέσο: SM1, χωρητικότητα: 387219 bytes, Τύπος: FlashDrive
USB stick Αποθηκευτικό μέσο: SM2, χωρητικότητα: 981156 bytes, Τύπος: FlashDrive
USB stick Αποθηκευτικό μέσο: SM3, χωρητικότητα: 755591 bytes, Τύπος: FlashDrive
USB stick Αποθηκευτικό μέσο: SM4, χωρητικότητα: 150761 bytes, Τύπος: FlashDrive
```

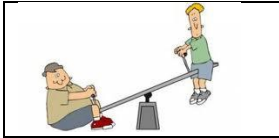
...

```
Αποθηκευτικό μέσο: SM96, χωρητικότητα: 386446 bytes, Τύπος: HD
Αποθηκευτικό μέσο: SM97, χωρητικότητα: 213386 bytes, Τύπος: HD
Αποθηκευτικό μέσο: SM98, χωρητικότητα: 154327 bytes, Τύπος: HD
Αποθηκευτικό μέσο: SM99, χωρητικότητα: 218552 bytes, Τύπος: HD
Αποθηκευτικό μέσο: SM100, χωρητικότητα: 358843 bytes, Τύπος: HD
Συνολική χωρητικότητα: 45234027
```

Υποδείξεις

- Για χρήση κλάσεων που αφορούν τυχαίους αριθμούς:
 - Θα χρειαστεί να κάνετε `import java.util.Random;`
 - If «`Random r = new Random();`» then «`r.nextInt(int bound)`» returns a random integer from 0 (inclusive) to bound (exclusive).
 - Για δημιουργία και εκτύπωση ενός τυχαίου αριθμού μεταξύ 0.1 και 1.0:
`double random = new Random().nextDouble();`
`System.out.println(random);`





Άσκηση 2. Τραμπάλα

Αξία: 20 μονάδες

Προδιαγραφή και υλοποίηση Κλάσεων, Εξαιρέσεις, Διεπαφές

Προσοχή (για τα παραδοτέα): Ακολουθήστε πιστά τη δομή του project που σας έχει δοθεί και συμπληρώστε καταλλήλως τα αρχεία βάσει της εκφώνησης που ακολουθεί. (Στο package ex02.a τοποθετείστε τη ζητούμενη τεκμηρίωση των υπογραφών του ADT (με ένα έγγραφο .doc ή .docx ή .pdf))

Καλείστε να σχεδιάσετε και να υλοποιήσετε έναν Αφαιρετικό Τύπο Δεδομένων ADT (Abstract Data Type) με το όνομα **Τραμπάλα**. Ο ADT πρέπει να επιτρέπει να προσθέτουμε ή να βγάζουμε πράγματα, στην αριστερή ή στη δεξιά μεριά της τραμπάλας, να ελέγχουμε εάν ένα πράγμα υπάρχει στην τραμπάλα, να ελέγχουμε εάν η τραμπάλα ισορροπεί, εάν είναι ακέραιη ή σπασμένη, και ποιο είναι το συνολικό βάρος των πραγμάτων που έχει πάνω της.

Συγκεκριμένα θεωρείστε ότι ένα *πράγμα* αποτελείται από ένα όνομα (String) και έναν ακέραιο (το βάρος του σε γραμμάρια). Επίσης κάθε τραμπάλα έχει (εκ κατασκευής) ένα όριο φόρτωσης (είναι στην ουσία η αντοχή του ξύλου), ας το πούμε K . Αν μια τραμπάλα ισορροπεί και το άθροισμα του βάρους που έχει στο δεξί και στο αριστερό της μέρος υπερβαίνει το όριο φόρτωσης K , τότε η τραμπάλα σπάει. Αν έχουμε μια τραμπάλα που στη μία της πλευρά έχει βάρος $2K$ τότε δεν σπάει (διότι ακουμπάει στο έδαφος). Αν όμως βάλουμε στην άλλη της πλευρά ένα βάρος $3K$ τότε στην πορεία της προς την άλλη πλευρά θα σπάσει.

α) [8 μονάδες]

Δώστε τις υπογραφές των μεθόδων του ADT Τραμπάλα (διακρίνοντας τις σε constructors, accessors, transformers, observers) και τις προ/μετα-συνθήκες (pre/post-conditions) κάθε μεθόδου (στα αγγλικά). Επίσης δώστε αναλλοίωτες συνθήκες (invariants) εκφράζοντας τις ως εξισωτικά αξιώματα (equational axioms στα σχόλια). Τα παραπάνω περιλαμβάνουν τον προσδιορισμό των εξαιρέσεων (exceptions) και που και πότε θα εγείρονται (ο τύπος του παραδοτέου περιγράφεται παρακάτω).

β) [7 μονάδες]

Δώστε μια υλοποίηση του ADT σε Java (μην λησμονήσετε να ορίσετε τις απαιτούμενες εξαιρέσεις).

γ) [5 μονάδες]

(i) Δώστε ένα παράδειγμα κώδικα που να χρησιμοποιεί την υλοποίηση της τραμπάλας και να δημιουργεί μια τραμπάλα την οποία φορτώνει τόσο ώστε να σπάσει.

(ii) Τροποποιήστε την υλοποίησή σας ώστε ένα πράγμα (όπως και η τραμπάλα) να υλοποιεί μια διεπαφή (**WeightedThing**), όπου ορίζεται η μέθοδος **int getVaros()**. Αυτό πρέπει να επιτρέψει να βάλουμε πάνω σε μια τραμπάλα, άλλες κ.ο.κ. Για να το δοκιμάσετε, φτιάξτε μία τραμπάλα η οποία να είναι φορτωμένη με δύο άλλες τραμπάλες. Οι τελευταίες δύο να είναι φορτωμένες με πράγματα αλλά να μην είναι σπασμένες, αλλά η βασική τραμπάλα να έχει σπάσει λόγω τους βάρους των δύο τραμπάλων.

Στην κονσόλα να τυπώνονται τα κατάλληλα μηνύματα που να κάνουν σαφές τι πετυχαίνει και τι όχι και γιατί.

Επίσης: Τέλος δοκιμάστε να φορτώσετε μια τραμπάλα στον εαυτό της. Τότε η υλοποίηση της μεθόδου που υπολογίζει το βάρος της πρέπει να κολλήσει και εν τέλει να λάβετε `StackOverflowError`.



Άσκηση 3. *Game of Thrones*: Ένας απλός Πυρήνας για Παιχνίδια Ρόλων (core for Role Playing Games)



Αξία: 40 μονάδες

Προδιαγραφή [και Υλοποίηση] Κλάσεων, Εξαιρέσεις, Διεπαφές

Bonus: (για την καλύτερη εργασία) Αναμνηστικός έπαινος

Προσοχή (για τα παραδοτέα): Ακολουθήστε πιστά τη δομή του project που σας έχει δοθεί και συμπληρώστε καταλλήλως τα αρχεία βάσει της εκφώνησης που ακολουθεί.

Σε αυτή την άσκηση καλείστε να μοντελοποιήσετε ένα τμήμα του περιβάλλοντος ενός παιχνιδιού ρόλων συγκεκριμένα του Game of Thrones. Σε αυτό το παιχνίδι εμφανίζονται διάφορες οντότητες, όπως **Soldier**, **Weapon**, **GreatHouse**.

Soldier	Weapon	GreatHouse	
		HouseStark	HouseLannister
			

Σύντομη Περιγραφή:

Κάθε μία οντότητα έχει τα δικά της χαρακτηριστικά π.χ. η οντότητα στρατιώτης (**Soldier**) έχει όνομα, επώνυμο και υγεία. Έχουμε δυο διαφορετικές κατηγορίες στρατιωτών (1) στρατιώτης-ξιφομάχος (**Swordman**) και (2) στρατιώτης-τοξότης (**archer**). Σε μία μάχη μπορούν να συμμετέχουν μόνο οντότητες που υλοποιούν (implements) τη διεπαφή (**Warrior**) δηλαδή μόνο οι στρατιώτες (**Soldier**).

Η κλάση στρατιώτης εξειδικεύει (extends) την κλάση **WeaponCarrier**, πράγμα που σημαίνει ότι κάθε στρατιώτης έχει τη δυνατότητα να κουβαλάει το δικό του όπλο (**Weapon**). Αυτό του δίνει την επιπλέον ικανότητα να κάνει μεγαλύτερη ζημιά στον αντίπαλο του κατά τη διάρκεια της μάχης. Κάθε όπλο (**weapon**) έχει τη δική του δύναμη (**power**) που όπως αναφέραμε παραπάνω σχετίζεται με την επιπλέον ζημιά που μπορεί να γίνει στο στρατιώτη που δέχεται την επίθεση. Συνεχίζοντας, η κλάση όπλο (**weapon**) χωρίζεται και αυτή με τη σειρά της σε δυο κατηγορίες (1) σπαθί (**Sword**) και (2) τόξο (**Arc**), ένα σπαθί μπορεί να χρησιμοποιηθεί μόνο από στρατιώτη-ξιφομάχο και αντίστοιχα ένα τόξο μόνο από στρατιώτη-τοξότη.

Τέλος αποτυπώνουμε την έννοια της βασιλικής οικογένειας με την κλάση **GreatHouse**. Κάθε βασιλική οικογένεια έχει το δικό της όνομα, έμβλημα, ρητό, άρχοντα όπως ακόμα και το δικό της στρατό. Στα πλαίσια της άσκησης θα προετοιμάσουμε τους στρατούς των δυο μεγάλων οικογενειών **HouseStark** και **HouseLannister** (επεκτείνουν την κλάση **GreatHouse**) για την τελική μάχη, ώστε στο τέλος μόνο μια να επικρατήσει και να ανέβει στο σιδερένιο θρόνο. Καλή μάχη!

Ακολουθεί Αναλυτική Περιγραφή, για ευκολία μπορείτε να ακολουθήσετε τα βήματα:

Βήμα 1: Υλοποιήστε την **abstract class** “**Weapon**”. Ένα “**Weapon**” έχει τα εξής γνωρίσματα:

- *power
- WeaponCarrier holder - Ο κάτοχος του όπλου (η κλάση θα υλοποιηθεί στο επόμενο βήμα “Βήμα 4”).



*Το χαρακτηριστικό **power** δε μπορεί να αλλάξει μετά από την αρχική απόδοση τιμής (Πρέπει να χαρακτηριστεί με το keyword final).

Το συμβόλαιο του ADT ενός **“Weapon”** είναι:

Όνομα μεθόδου	Περιγραφή μεθόδου
Weapon(int power)	Δημιουργεί ένα νέο weapon με δύναμη και χωρίς κάτοχο.
int getPower()	Επιστρέφει τη δύναμη του weapon.
void setHolder(WeaponCarrier holder)	Αναθέτει τον κάτοχο του όπλου.
WeaponCarrier getHolder()	Επιστρέφει τον κάτοχο του weapon, αν το weapon δεν έχει κάτοχο επιστρέφει null.
abstract String toString()	Επιστρέφει τις πληροφορίες για ένα weapon.

Βήμα 2: Υλοποιήστε την **class “Sword”** η οποία επεκτείνει την κλάση weapon. Ένα “Sword” ΔΕΝ έχει επιπλέον γνωρίσματα.

Το συμβόλαιο του ADT ενός **“Sword”** είναι:

Όνομα μεθόδου	Περιγραφή μεθόδου
Sword()	Δημιουργεί ένα νέο sword με τυχαία δύναμη (παίρνει τιμές από [3, 4]).
Sword(int power)	Δημιουργεί ένα νέο sword με δύναμη. Περιορισμοί: <ul style="list-style-type: none">• Η δύναμη του sword παίρνει τιμές [3, 4].
String toString()	Επιστρέφει τις πληροφορίες για τη δύναμη και τον κάτοχο του sword. Πχ “The sword has power 10 and is owned by Eddard Stark.”

Βήμα 3: Υλοποιήστε την **class “Arc”** η οποία επεκτείνει την κλάση weapon. Ένα “Arc” ΔΕΝ έχει επιπλέον γνωρίσματα.

Το συμβόλαιο του ADT ενός **“Arc”** είναι:

Όνομα μεθόδου	Περιγραφή μεθόδου
Arc()	Δημιουργεί ένα νέο arc με τυχαία δύναμη (παίρνει τιμές από [1, 2]) , χωρίς κάτοχο.

Arc(int power)	Δημιουργεί ένα νέο arc με δύναμη και χωρίς κάτοχο. Περιορισμοί: <ul style="list-style-type: none"> Η δύναμη του arc παίρνει τιμές [1, 2].
String toString()	Επιστρέφει τις πληροφορίες για τη δύναμη και τον κάτοχο του arc. Πχ "The arc has power 5 and is owned by Tywin Lannister."

Βήμα 4: Υλοποιήστε την **abstract class "WeaponCarrier"**. Ένας "WeaponCarrier" έχει το γνώρισμα:

- Weapon weapon – ένα όπλο

Το συμβόλαιο του ADT ενός **"WeaponCarrier"** είναι:

Όνομα μεθόδου	Περιγραφή μεθόδου
WeaponCarrier()	Δημιουργεί ένα νέο weaponCarrier, ο οποίος δεν έχει ακόμα στην κατοχή κάποιο όπλο (weapon = null).
Weapon getWeapon()	Επιστρέφει το όπλο του WeaponCarrier.
abstract void setWeapon(Weapon weapon)	Αναθέτει στον WeaponCarrier ένα όπλο.
boolean hasWeapon()	Επιστρέφει true αν ο WeaponCarrier έχει όπλο.

Βήμα 5: Υλοποιήστε το **interface "Warrior"**. Το συμβόλαιο του ADT ενός "Warrior" είναι:

Όνομα μεθόδου	Περιγραφή μεθόδου
void attack(Warrior adversary)	Επιτίθεται σε ένα αντίπαλο. Περιορισμοί: <ol style="list-style-type: none"> Ένας warrior δε μπορεί να επιτεθεί στον εαυτό του. Ο warrior που κάνει την επίθεση πρέπει να έχει health >0. Στο τέλος της επίθεσης πρέπει να τυπώνεται ότι ο τρέχον warrior (το όνομα του) επιτέθηκε στον adversary (το όνομα του) καθώς και την νέα κατάσταση (health) του adversary. Μετά από την επίθεση, η κατάσταση του adversary πρέπει να είναι μικρότερη από την προηγούμενη του κατάσταση. Αλγόριθμος (υπολογισμού της δύναμης που θα αφαιρεθεί από τον adversary) <ol style="list-style-type: none"> Παίρνουμε τη δύναμη του επιτιθέμενου. Αν ο επιτιθέμενος έχει όπλο, παίρνουμε τη δύναμη του όπλου. Το άθροισμα των βημάτων 1, 2 είναι η συνολική μείωση που θα γίνει στην κατάσταση του αμυνόμενου.

boolean isDefeated()	Επιστρέφει αν ο warrior έχει ηττηθεί (δηλαδή το health είναι ≤ 0).
int getHealthCondition()	Επιστρέφει την κατάσταση του warrior.
void setHealthCondition(int condition)	Αναθέτει την κατάσταση στον warrior. Περιορισμοί: <ul style="list-style-type: none"> Ο warrior δεν πρέπει να έχει ηττηθεί.
int getPower()	Επιστρέφει τη δύναμη για επίθεση που έχει ο warrior
String getCallSign()	Επιστρέφει το ονοματεπώνυμο του warrior.

Βήμα 6: Υλοποιήστε την **abstract class “Soldier”** η οποία επεκτείνει την κλάση **weaponCarrier** και υλοποιεί το interface **Warrior**. Ένας **“Soldier”** έχει τα εξής γνωρίσματα:

- *firstName - Δεν πρέπει να είναι κενό
- *lastName - Δεν πρέπει να είναι κενό
- health - Η υγεία του στρατιώτη. Κατά τη διάρκεια της μάχης μπορεί να πάρει αρνητική τιμή, για οποιαδήποτε τιμή $health \leq 0$ θα σημαίνει ότι ο soldier έχει σκοτωθεί)
- *power

*Τα χαρακτηριστικά firstName, lastName, power δε μπορούν να αλλάξουν μετά από την αρχική απόδοση τιμών. (Πρέπει να χαρακτηριστούν με το keyword **final**.)

Το συμβόλαιο του ADT ενός **“Soldier”** είναι:

Όνομα μεθόδου	Περιγραφή μεθόδου
Soldier (String firstName, String lastName, int health, int power)	Δημιουργεί ένα νέο soldier με ονοματεπώνυμο, υγεία και δύναμη. Περιορισμοί: <ul style="list-style-type: none"> Το όνομα και το επώνυμο δεν πρέπει να είναι κενά.
void attack(Warrior adversary)	Επιτίθεται σε ένα αντίπαλο. Περιορισμοί: <ul style="list-style-type: none"> Προσοχή στους περιορισμούς που περιγράφονται στο βήμα 5.
int getHealthCondition()	Επιστρέφει την κατάσταση του soldier.
void setHealthCondition(int condition)	Αναθέτει την κατάσταση στον soldier. Περιορισμοί: <ul style="list-style-type: none"> Ο soldier δεν πρέπει να έχει ηττηθεί.
boolean isDefeated()	Επιστρέφει true αν ο soldier έχει σκοτωθεί (όταν δηλαδή η τιμή του $health \leq 0$), αλλιώς επιστρέφει false.

String getCallSign()	Επιστρέφει το ονοματεπώνυμο του soldier.
int getPower()	Επιστρέφει τη δύναμη για επίθεση που έχει ο soldier
abstract String toString()	Επιστρέφει όλες τις πληροφορίες για ένα soldier.

Βήμα 7: Υλοποιήστε την class “**Swordman**” η οποία επεκτείνει την κλάση **Solider**. Ένας “Swordman” ΔΕΝ έχει επιπλέον γνωρίσματα.

Το συμβόλαιο του ADT ενός “**Swordman**” είναι:

Όνομα μεθόδου	Περιγραφή μεθόδου
Swordman (String firstName, String lastName)	Δημιουργεί ένα νέο swordman με ονοματεπώνυμο, τυχαία υγεία (παίρνει τιμές από [5, 10]) και default δύναμη ίση με 4.
Swordman (String firstName, String lastName, int health)	Δημιουργεί ένα νέο swordman με ονοματεπώνυμο, υγεία και default δύναμη ίση με 4. Περιορισμοί: <ul style="list-style-type: none"> • Η υγεία του swordman παίρνει τιμές [5, 10].
void setWeapon(Weapon weapon)	Αναθέτει στον WeaponCarrier ένα όπλο. Περιορισμοί: <ol style="list-style-type: none"> 1. Το weapon δε μπορεί να είναι null. 2. Το weapon πρέπει να είναι στιγμιότυπο της κλάσης “Sword”. Βοήθεια: Για να βεβαιωθείτε ότι το αντικείμενο είναι τύπου “Sword” δείτε πως μπορείτε να χρησιμοποιήσετε το instanceof. Προσοχή στην ανανέωση του πεδίου holder του Weapon.
String toString()	Επιστρέφει όλες τις πληροφορίες για ένα Swordman. Πχ “The swordman Eddard Stark has power 4 and health 5.”

Βήμα 8: Υλοποιήστε την class “**Archer**”, η οποία επεκτείνει την κλάση **Solider**. Ένας “Archer” ΔΕΝ έχει επιπλέον γνωρίσματα.

Το συμβόλαιο του ADT ενός “**Archer**” είναι:

Όνομα μεθόδου	Περιγραφή μεθόδου
Archer (String firstName, String lastName)	Δημιουργεί ένα νέο archer με ονοματεπώνυμο, τυχαία υγεία (παίρνει τιμές από [1, 5]) και default δύναμη ίση με 2.



Archer (String firstName, String lastName, int health)	Δημιουργεί ένα νέο archer με ονοματεπώνυμο, υγεία και default δύναμη ίση με 2. Περιορισμοί: <ul style="list-style-type: none"> • Η υγεία του archer παίρνει τιμές [1, 5].
void setWeapon(Weapon weapon)	Αναθέτει στον WeaponCarrier ένα όπλο. Περιορισμοί: <ol style="list-style-type: none"> 1. Το weapon δε μπορεί να είναι null. 2. Το weapon πρέπει να είναι στιγμιότυπο της κλάσης "Arc". Βοήθεια: Για να βεβαιωθείτε ότι το αντικείμενο είναι τύπου "Arc" δείτε πως μπορείτε να χρησιμοποιήσετε το instanceof. Προσοχή στην ανανέωση του πεδίου holder του Weapon.
String toString()	Επιστρέφει όλες τις πληροφορίες για ένα Archer. Πχ "The archer Tywin Lannister has power 2 and health 1."

Βήμα 9: Υλοποιήστε την abstract class **"GreatHouse"**. Ένα "GreatHouse" έχει τα εξής γνωρίσματα:

- *name – Δεν πρέπει να είναι κενό
- *sigil – Δεν πρέπει να είναι κενό
- *words – Δεν πρέπει να είναι κενό
- *lord – Δεν πρέπει να είναι κενό
- *army – πίνακας που θα κρατάει τους soldiers που έχει στη διάθεση του αυτό το GreatHouse

*Τα χαρακτηριστικά **name**, **sigil**, **words**, **lord** δε μπορούν να αλλάξουν μετά από την αρχική απόδοση τιμών (πρέπει να χαρακτηριστούν με το keyword **final**). Αξίζει να σημειωθεί ότι το γνώρισμα **army** μπορεί να χαρακτηριστεί και αυτό **final**. Αυτό σημαίνει ότι μετά την αρχική ανάθεση στο army ενός στιγμιότυπου πχ ArrayList δεν υπάρχει η δυνατότητα να αναθέσουμε στη συνέχεια κάποιο άλλο στιγμιότυπο τύπου ArrayList. Παρόλα αυτά το περιεχόμενο του ArrayList μπορεί να μεταβάλετε. Δείτε τον παρακάτω σύνδεσμο <https://stackoverflow.com/questions/10750791/what-is-the-sense-of-final-arraylist>.

Το συμβόλαιο του ADT ενός **"GreatHouse"** είναι:

Όνομα μεθόδου	Περιγραφή μεθόδου
GreatHouse (String name, String sigil, String words, String lord)	Δημιουργεί ένα νέο greatHouse με όνομα, έμβλημα, ρητό, άρχοντα. Σε αυτή την περίπτωση θα πρέπει να δημιουργηθεί νέο στιγμιότυπο για τον πίνακα army χωρίς όμως να έχει στρατιώτες μέσα.



	Περιορισμοί: <ul style="list-style-type: none"> Τα name, sigil, words, lord δεν πρέπει να είναι κενά.
void addSoldier(Solider soldier)	Προσθέτει ένα νέο στρατιώτη στο στρατό του greatHouse. Περιορισμοί: <ul style="list-style-type: none"> Ο στρατιώτης δεν πρέπει να είναι null.
Soldier getSoldier()	Επιστρέφει ένα στρατιώτη από το από το στρατό. Περιορισμοί: <ul style="list-style-type: none"> Ο στρατιώτης πρέπει να είναι ζωντανός
boolean isDefeated()	Επιστρέφει true αν το greatHouse δεν έχει στη διάθεση του άλλους ζωντανούς στρατιώτες για να πολεμήσουν, αλλιώς false. Στην περίπτωση που επιστρέψει true σημαίνει ότι το greatHouse έχασε την μάχη και τον πόλεμο.
String toString()	Επιστρέφει όλες τις πληροφορίες για ένα greatHouse. Πχ "The GreateHouse of Starks has lord Eddard Stark. Their sigil is a 'A grey direwolf on a white field' and their words are 'Winter Is Coming'."

Βήμα 10: Για την ολοκλήρωση της άσκησης θέλουμε να φτιάξουμε ακόμα δυο classes την **"HouseStark"** και την **"HouseLannister"** και οι δυο επεκτείνουν την κλάση **GreatHouse**. Αυτές οι δύο κλάσεις θα ανήκουν στην κατηγορία των **Singleton Classes**, πράγμα που σημαίνει ότι θα επιτρέψουμε τη δημιουργία μόνο ενός στιγμιότυπου για κάθε μία. Μπορείτε να δείτε στο παρακάτω κομμάτι κώδικα, πως φτιάχνουμε μια singleton class. Για περισσότερες λεπτομέρειες δείτε τον σύνδεσμο https://www.tutorialspoint.com/java/java_using_singleton.htm .

```
// File Name: ClassicSingleton.java
public class ClassicSingleton {

    private static ClassicSingleton instance = null;

    // Private constructor!
    private ClassicSingleton() {
        // Exists only to defeat instantiation.
    }

    // If not static it would be useless
    public static ClassicSingleton getInstance() {
        if(instance == null) {
            instance = new ClassicSingleton();
        }
        return instance;
    }

    /* A simple method*/
    public static void demoMethod() {
```

```

        System.out.println("demoMethod for singleton");
    }
}

// File Name: SingletonDemo.java
public class SingletonDemo {

    public static void main(String[] args) {
        ClassicSingleton tmp = ClassicSingleton.getInstance( );
        tmp.demoMethod( );
    }
}

```

Βήμα 11: Υλοποιήστε την class “**HouseStark**”. Το “HouseStark” ΔΕΝ έχει επιπλέον γνωρίσματα. Το “houseStark” θα είναι μια Singleton Class όπως περιγράψαμε και παραπάνω.

Το συμβόλαιο του ADT ενός “**HouseStark**” είναι:

Όνομα μεθόδου	Περιγραφή μεθόδου
HouseStark (String name, String sigil, String words, String lord)	<p>Δημιουργεί το “μοναδικό” στιγμιότυπο της houseStark.</p> <p>Για αρχικές τιμές μπορείτε να δώσετε τα παρακάτω:</p> <ul style="list-style-type: none"> • name = “Stark” • sigil = “A grey direwolf on a white field” • words = “Winter Is Coming” • lord = “Eddard Stark”

Βήμα 12: Υλοποιήστε την class “**HouseLannister**”. Το “HouseLannister” ΔΕΝ έχει επιπλέον γνωρίσματα. Το “houseLannister” θα είναι μια Singleton Class όπως περιγράψαμε και παραπάνω.

Το συμβόλαιο του ADT ενός “**HouseLannister**” είναι:

Όνομα μεθόδου	Περιγραφή μεθόδου
HouseLannister (String name, String sigil, String words, String lord)	<p>Δημιουργεί το “μοναδικό” στιγμιότυπο της houseLannister.</p> <p>Για αρχικές τιμές μπορείτε να δώσετε τα παρακάτω:</p> <ul style="list-style-type: none"> • name = “Lannister” • sigil = “A golden lion rampant on a crimson field” • words = “A Lannister Always Pays His Debts” • lord = “Tywin Lannister”

Βήμα 13: Υλοποιείτε τα exceptions DeadWarriorException (χρησιμοποιείτε το σε περιπτώσεις όπου ο στρατιώτης δεν πρέπει να είναι νεκρός), ακόμα το WarriorAttackHissselfException (χρησιμοποιείτε το όταν ο στρατιώτης επιτίθεται στον εαυτό του). Πέρα από τα exceptions που ζητήθηκαν, είστε ελεύθεροι να χρησιμοποιήσετε επιπλέον exceptions σε όποια άλλα σημεία κρίνετε ότι χρειάζονται (εμπλουτίζοντας καταλλήλως και τις υπογραφές των αντίστοιχων μεθόδων). Τέλος σας δίνεται (στο παράρτημα) μία main μέθοδος (θα βρείτε το αντίστοιχο αρχείο και στο moodle), η

οποία θα πρέπει να τρέχει (και να τερματίζει) με τον κώδικά σας, χωρίς λάθη, καθώς και μία ενδεικτική έξοδος αυτής (το αντίστοιχο αρχείο θα το βρείτε και στο moodle).

Προσοχή η ενδεικτική έξοδος αλλάζει σε κάθε τρέξιμο λόγω των random που χρησιμοποιούνται.



Άσκηση 4. BusTicketSeller Machine

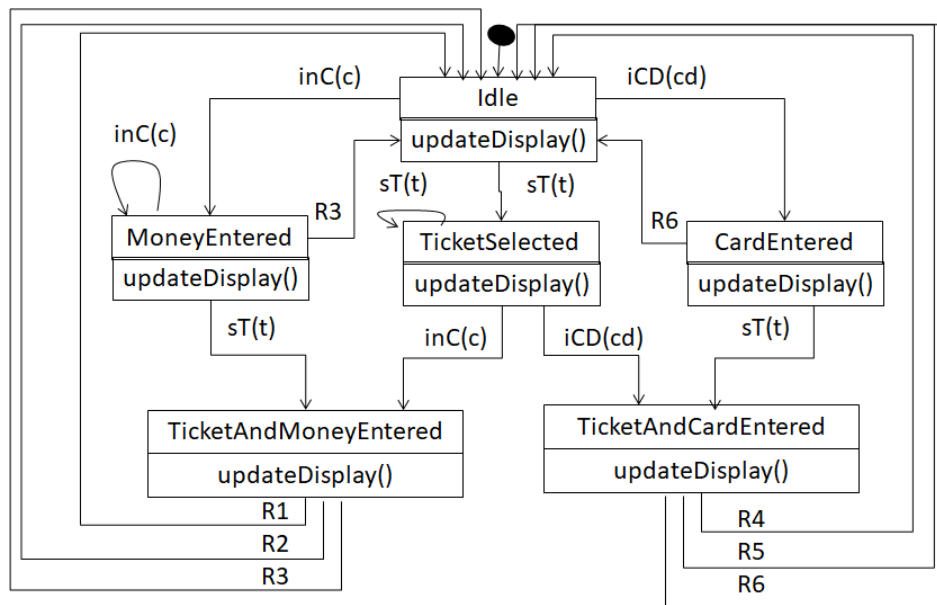
Αξία: 25 μονάδες

Προδιαγραφή [και Υλοποίηση] Κλάσεων, Εξαιρέσεις, Διεπαφές

Προσοχή (για τα παραδοτέα): Ακολουθήστε πιστά τη δομή του project που σας έχει δοθεί και συμπληρώστε καταλλήλως τα αρχεία βάσει της εκφώνησης που ακολουθεί.

Καλείστε να υλοποιήσετε έναν αφαιρετικό τύπο **BusTicketsSellerADT** (και εν συνεχεία μία κλάση **BusTicketsSeller**) η οποία θα υλοποιεί τη λογική ενός αυτόματου μηχανήματος πώλησης εισιτηρίων λεωφορείων. Ακολουθεί το **Διάγραμμα Καταστάσεων** (UML State Diagram) κάθε στιγμιότυπου του BusTicketsSellerADT. Τα κουτιά υποδηλώνουν τις καταστάσεις ενώ οι ακμές εκφράζουν τις μεταβάσεις μεταξύ των καταστάσεων, ανάλογα με τα συμβάντα (που αποτελούν τις ετικέτες των ακμών). Τα συμβάντα ακολουθούν τον κανόνα 'συμβάν'(παράμετροι) [συνθήκες που πρέπει να ισχύουν] / 'ενέργεια' όπου μόνο το πρώτο μέρος (δηλαδή το συμβάν) είναι υποχρεωτικό. Μπορείτε να διαβάσετε παραπάνω και να δείτε παραδείγματα διαγραμμάτων καταστάσεων στους παρακάτω συνδέσμους:

- http://www.sparxsystems.com/resources/uml2_tutorial/uml2_statediagram.html
- https://en.wikipedia.org/wiki/UML_state_machine



Συντομογραφίες:

sT (t): selectTicket (Ticket t);

inC (c): insertCoin (Coin c);

iCD (cd): inserCard (Card cd);

R1: finish()[enoughCoins AND ticketInStock]/ {returnChange(); deliverTicket()}

R2: finish()[!enoughCoins OR !ticketInStock]/ returnMoney()

R3: cancel()/ returnCoins()



```
R4: finish()[enoughMoneyInCard AND ticketInStock]/ {withdrawCostFromCard (); takeOutCard(); deliverTicket()}
R5: finish()[!enoughMoneyInCard OR ticketInStock]/ takeOutCard();
R6: cancel()/ takeOutCard()
```

Εικόνα: Το διάγραμμα καταστάσεων ενός BusTicketSeller

Περιγραφή

Στην κατάσταση 'IDLE' ένας BusTicketSeller θα αναμένει είσοδο από το χρήστη, η οποία θα δίνεται μέσω της επιλογής εισιτηρίου, μέσω της εισαγωγής χρημάτων ή μέσω της εισαγωγής κάρτας. Τα εισιτήρια θα είναι τύπου 'Ticket' με τιμές: 'Ολόκληρο' (αξία 2,0€), 'Φοιτητικό' (αξία 1,5€), και 'Πολυτεχνικό' (αξία 1,0€), τα χρήματα τύπου 'Coin' με τιμές: **0,05€, 0,1€, 0,2€, 0,5€, 1€ και 2€**, ακόμα θα υπάρχει και ο τύπος κάρτα 'Card'. Τα αντικείμενα τύπου 'Ticket' θα πρέπει να έχουν ένα γνώρισμα 'price', που θα αντιστοιχεί στην τιμή αγοράς του εκάστοτε εισιτηρίου όπου: το 'Ολόκληρο' έχει αξία 2,0€, το 'Φοιτητικό' έχει αξία 1,5€, και το 'Πολυτεχνικό' έχει αξία 1,0€. Το BusTicketSeller θα πρέπει να έχει μια τιμή 'stock' η οποία θα υποδεικνύει το συνολικό αριθμό εισιτηρίων που υπάρχουν σε απόθεμα μέσα στο μηχανήμα πώλησης (θεωρούμε ότι το μηχανήμα έχει ένα κοινό stock για όλους τους τύπους εισιτηρίων).

Ξεκινώντας από την κατάσταση 'Idle', επιλέγοντας κάποιο εισιτήριο θα πηγαίνει στην κατάσταση 'TicketSelected'. Αν είχε πρώτα βάλει χρήματα θα πηγαίνει στην κατάσταση 'MoneyEntered'. Αν έχει κάνει και τα δύο, σε οποιαδήποτε σειρά, θα φτάσει στην κατάσταση 'TicketAndMoneyEntered'. Αντίστοιχα, επιλέγοντας κάποιο εισιτήριο και εισάγοντας κάποια κάρτα (ή στην αντίθετη σειρά), θα μεταφέρεται στην κατάσταση 'TicketAndCardEntered'. Από την κατάσταση 'TicketAndMoneyEntered' (ή αντίστοιχα από την 'TicketAndCardEntered'), μέσω του συμβάντος 'finished' θα γίνεται έλεγχος αν το χρηματικό ποσό που έχει εισαχθεί (ή αντίστοιχα το υπόλοιπο της κάρτας) είναι ίσο ή μεγαλύτερο από την τιμή ('price') του εισιτηρίου, καθώς και αν υπάρχει απόθεμα εισιτηρίων. Αν οι δυο αυτές προϋποθέσεις πληρούνται, τότε θα επιστρέφονται στο χρήστη τα ρέστα (αν χρειάζεται) μέσω της returnChange() (ή θα αφαιρείται το κόστος του εισιτηρίου από την κάρτα, μέσω της withdrawMoneyFromCard()), θα μειώνεται το απόθεμα των εισιτηρίων, θα παραδίνεται το εισιτήριο στον χρήστη μέσω της deliverTicket() και ο χρήστης θα μεταφέρεται ξανά πίσω στην κατάσταση 'IDLE'. Οι ακριβείς μεταβάσεις είναι αυτές που προσδιορίζονται στο διάγραμμα καταστάσεων.

Σημείωση:

Σε κάθε κατάσταση, θα πρέπει να καλείτε την updateDisplay(), η οποία θα τυπώνει πληροφορίες στο output σχετικά με την κατάσταση στην οποία βρισκόμαστε, αν και ποιο εισιτήριο είναι επιλεγμένο, την τιμή του, το συνολικό απόθεμα εισιτηρίων, τα χρήματα που έχουν εισαχθεί και τα ρέστα που θα πρέπει να επιστραφούν στον χρήστη.

Ζητούμενα

α) [3 μονάδες]

Κάντε την προδιαγραφή του BusTicketsSeller_ADT θεωρώντας τα συμβάντα ως μεθόδους του ADT. Φροντίστε ώστε η προδιαγραφή να είναι σύμφωνη με το διάγραμμα καταστάσεων, και άρα η προδιαγραφή των μεταβάσεων να είναι σύμφωνη με το διάγραμμα καταστάσεων. Επιπλέον, θα πρέπει να προσφέρεται μέθοδος (getState()) η οποία θα επιστρέφει την κατάσταση ενός αντικειμένου (ενός στιγμιότυπου της BusTicketsSeller). Για την παράσταση των καταστάσεων, χρησιμοποιήστε έναν enum.

Συγκεκριμένα, η προδιαγραφή πρέπει να είναι εκφρασμένη ως ένα Java Interface (public interface **BusTicketsSeller_ADT** {...}). Δώστε τις υπογραφές (signatures) των προβλεπόμενων



λειτουργιών του, το είδος τους ανάλογα με την επίδραση που έχουν στην κατάσταση των αντικειμένων (constructors, observers, accessors, transformers), καθώς και τις προ/μετά συνθήκες και τις αμετάβλητες συνθήκες (preconditions, postconditions, invariants). Επίσης ορίστε τις εξαιρέσεις (Exceptions) που απαιτούνται. Τεκμηριώστε πλήρως την προδιαγραφή σας με χρήση Javadocs. Πλοηγηθείτε με ένα πρόγραμμα περιήγησης ιστού για να δείτε αν η παραχθείσα τεκμηρίωση είναι εντάξει.

β) [6 μονάδες]

Υλοποιείστε την κλάση **BusTicketsSeller** και τις μεθόδους της, βάσει της προδιαγραφής του **BusTicketsSeller_ADT** που κάνατε στο σκέλος (α) και ό,τι άλλο χρειάζεται για να λειτουργεί σωστά.

γ) [10 μονάδες]

Προσθέστε κώδικα για τον έλεγχο της ορθότητας της υλοποίησης του **BusTicketsSeller** (μέσα στο αρχείο **BusTicketSellerTest**), δηλαδή να ελέγχει αν η υλοποίηση που δώσατε στο σκέλος (β) ακολουθεί το διάγραμμα καταστάσεων.

δ) [6 μονάδες]

Χρησιμοποιείστε τον ακόλουθο κώδικα που σας δίνεται (στο παράρτημα) ο οποίος δημιουργεί ένα γραφικό περιβάλλον το οποίο προσομοιώνει τη φυσική υπόσταση μιας μηχανής πώλησης εισιτηρίων. Στην ουσία ο κώδικας που σας δίνεται μεσολαβεί μεταξύ του χρήστη και του κώδικα που φτιάξατε στα σκέλη (α) και (β). Ο κώδικας που φτιάξατε στα (α) και (β) είναι ο πυρήνας της λειτουργικότητας (το model κομμάτι της τελικής εφαρμογής). Οι κλάσεις που σας δίνονται είναι στην ουσία πελάτες (clients) της κλάσης **BusTicketsSeller** που φτιάξατε (θα καλούν μεθόδους που έχετε ορίσει στο interface **BusTicketsSeller_ADT**). Το γραφικό δίνει τον τρόπο για να πραγματοποιηθούν τα συμβάντα (άρα οι κλήσεις προς τις μεθόδους που φτιάξατε).

Ο κώδικας του **APPILCATION.java** που σας δίνεται πρέπει να μπορεί να τρέχει ως έχει σωστά και χωρίς κανένα λάθος. Εκτός της συμπλήρωσης για τα άλλα νομίσματα και τύπους εισιτηρίων, είστε ελεύθεροι (και συστήνεται) να βελτιώσετε τη γραφική διεπαφή (για να εξοικειωθείτε με τις γραφικές διεπαφές που θα σας χρειαστούν στο project του μαθήματος).

Ο κώδικας για τη γραφική διεπαφή που σας δίνεται υπάρχει στο **elearn**.

Παράρτημα για την Άσκηση 3

Κώδικας δοκιμής:

```
package got;

import greatHouse.HouseLannister;
import greatHouse.HouseStark;
import java.util.Random;
import weapon.Arc;
import weapon.Sword;
import weaponCarrier.Archer;
import weaponCarrier.Soldier;
import weaponCarrier.Swordman;
```



```

/**
 * Game of Thrones (core for role playing games)
 *
 * @author George Nikitakis <nikitak at csd.uoc.gr>
 */
public class GoT {

    public static int getRandomVal(int min, int max) {
        Random rand = new Random();
        int n = rand.nextInt(max - min + 1) + min;

        return n;
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        HouseStark houseStark = null;
        HouseLannister houseLannister = null;

        try {
            //Get the instances of two houses
            houseStark = HouseStark.getInstance();
            houseLannister = HouseLannister.getInstance();

            //Construct Swordmen and Swords
            Swordman swordman_1 = new Swordman("Eddard", "Stark", 10);
            Swordman swordman_2 = new Swordman("Robb", "Stark");
            Swordman swordman_3 = new Swordman("Tywin", "Lannister", 10);
            Swordman swordman_4 = new Swordman("Jaime", "Lannister");

            Sword sword_1 = new Sword(4);
            Sword sword_2 = new Sword();

            //Swordmen get their swords
            swordman_1.setWeapon(sword_1);
            swordman_3.setWeapon(sword_2);

            //Construct Archers and Arcs
            Archer archer_1 = new Archer("Arya", "Stark", 4);
            Archer archer_2 = new Archer("Tyrion", "Lannister");

            Arc arc_1 = new Arc(1);
            Arc arc_2 = new Arc();

            //Archers get their arcs
            archer_1.setWeapon(arc_1);
            archer_2.setWeapon(arc_2);

            //Add some swordmen and archers to houseStark army
            houseStark.addSoldier(swordman_1);
            houseStark.addSoldier(swordman_2);
            houseStark.addSoldier(archer_1);

```

```

//Add some swordmen and archers to houseLannister army
houseLannister.addSoldier(swordman_3);
houseLannister.addSoldier(swordman_4);
houseLannister.addSoldier(archer_2);

Soldier soldierStark = null;
Soldier soldierLannister = null;

while (!houseStark.isDefeated() && !houseLannister.isDefeated()) {

    if (soldierStark == null || soldierStark.isDefeated()) {
        if (soldierStark != null) {
            System.out.println("Stark Army: " + soldierStark.getCallSign() + " died in the battlefield.");
        }

        soldierStark = houseStark.getSoldier();
        System.out.println("Stark Army: " + soldierStark.toString() + " Entering the battlefield!");
        if (soldierStark.hasWeapon()) {
            System.out.println("\tWeapon info: " + soldierStark.getWeapon().toString());
        }
        System.out.println("\n");
    }

    if (soldierLannister == null || soldierLannister.isDefeated()) {
        if (soldierLannister != null) {
            System.out.println("Lannister Army: " + soldierLannister.getCallSign() + " died in the battlefield.");
        }

        soldierLannister = houseLannister.getSoldier();
        System.out.println("Lannister Army: " + soldierLannister.toString() + " Entering the battlefield!");
        if (soldierLannister.hasWeapon()) {
            System.out.println("\tWeapon info: " + soldierLannister.getWeapon().toString());
        }
        System.out.println("\n");
    }

    int turn = getRandomVal(0, 1);
    if (turn == 0) {
        //Stark's soldier starts first!
        soldierStark.attack(soldierLannister);
        if (!soldierLannister.isDefeated()) {
            soldierLannister.attack(soldierStark);
        }
    } else {
        //Lannisters's soldier starts first!
        soldierLannister.attack(soldierStark);
        if (!soldierStark.isDefeated()) {
            soldierStark.attack(soldierLannister);
        }
    }
}

} catch (Exception e) {
    System.out.println(e);
}

```

```

    } finally {
        System.out.println(houseStark.isDefeated() ? houseLannister.toString() : houseStark.toString());
        System.out.println("Won the Iron Throne!");
    }
}
}

```

Ενδεικτική έξοδος:

Stark Army: The swordman Eddard Stark has power 4 and health 10. Entering the battlefield!

Weapon info: The sword has power '4' and is owned by 'The swordman Eddard Stark has power 4 and health 10'.

Lannister Army: The swordman Tywin Lannister has power 4 and health 10. Entering the battlefield!

Weapon info: The sword has power '4' and is owned by 'The swordman Tywin Lannister has power 4 and health 10'.

Eddard Stark attacked Tywin Lannister

Tywin Lannister healthPwr: 2

Tywin Lannister attacked Eddard Stark

Eddard Stark healthPwr: 2

Eddard Stark attacked Tywin Lannister

Tywin Lannister healthPwr: -6

Lannister Army: Tywin Lannister died in the battlefield.

Lannister Army: The swordman Jaime Lannister has power 4 and health 5. Entering the battlefield!

Eddard Stark attacked Jaime Lannister

Jaime Lannister healthPwr: -3

Lannister Army: Jaime Lannister died in the battlefield.

Lannister Army: The archer Tyrion Lannister has power 2 and health 4. Entering the battlefield!

Weapon info: The arc has power '2' and is owned by 'The archer Tyrion Lannister has power 2 and health 4'.

Eddard Stark attacked Tyrion Lannister

Tyrion Lannister healthPwr: -4

The GreateHouse of Stark, has lord Eddard Stark. Their sigil is 'A grey direwolf on a white field' and their words are 'Winter Is Coming'.

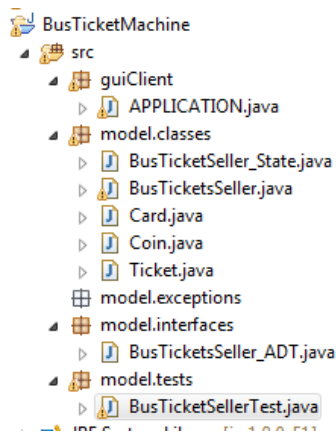
Won the Iron Throne!

Παράρτημα για την Άσκηση 4.δ

Το αντίστοιχο κομμάτι που σας δίνεται έχει την εξής δομή:



Yannis Tzitzikas, Department of Computer Science, University of Crete



```
package ex04.guiClient;

import java.awt.Color;
import java.awt.GridLayout;
import java.awt.Image;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.ImageIcon;

import ex04.model.classes.BusTicketsSeller;
import ex04.model.classes.Coin;
import ex04.model.classes.Ticket;
import ex04.model.interfaces.BusTicketsSeller_ADT;

/*
 * The class APPLICATION simulates (through a GUI) the physical part of a bus ticket selling machine.
 * It sends and receives messages to the Model part of the application.
 * (the model part of the application does not depend on the gui).
 */
// CoinButton can hold a reference to a Coin
// We define it as a class for creating buttons for several coins
class CoinButton extends JButton implements ActionListener {

    Coin c;

    @Override
    public void actionPerformed(java.awt.event.ActionEvent e) { // called on click
        System.out.println(c);
        APPLICATION.bts.insertCoin(c); // calls the corresponding method of the model
        APPLICATION.updateDisplay(); // updates the display
    }

    CoinButton(Coin c, String image) {
        super(c.toString());
        this.c = c;
        this.addActionListener(this); // for managing itself the clicks

        //Add the image to the button
        Image newimg = new ImageIcon(image).getImage().getScaledInstance(
            30, 30, java.awt.Image.SCALE_SMOOTH);
        this.setIcon(new ImageIcon(newimg));
    }
}
```

```

}

//TicketButton can hold a reference to a Ticket
//We define it as a class for creating buttons for several kinds of tickets
class TicketButton extends JButton implements ActionListener {

    Ticket t;

    public void actionPerformed(java.awt.event.ActionEvent e) {
        System.out.println(t);
        APPLICATION.bts.selectTicket(t); // calls the corresponding method of the model
        APPLICATION.updateDisplay();
    }

    TicketButton(Ticket t) {
        super(t.toString());
        this.t = t;
        this.addActionListener(this);
    }
}

// This the class with the main that starts the application
public class APPLICATION extends JFrame {

    static final String appName = "BUS TICKET SELLER v0.1 Application";

    static BusTicketsSeller_ADT bts; // only one instance is needed

    static CoinButton coin2Euro, coin10cents; // coin buttons
    static TicketButton ticketStudent; // ticketStudent
    static JLabel displayArea; // for displaying the status
    static JButton cancelButton; // the cancel button
    static JButton finishButton; // the finish button

    // it creates all coin buttons
    void createCoinButtons() {
        coin2Euro = new CoinButton(new Coin("2 Euro", 2F), "./resources/images/Two_euro.jpg");
        add(coin2Euro);
        coin10cents = new CoinButton(new Coin("10 cents", 0.1F), "./resources/images/Ten_cent.jpg");
        add(coin10cents);
        // students could complete this with the rest coins
    }

    // it creates all ticket buttons
    void createTicketButtons() {
        ticketStudent = new TicketButton(new Ticket("Foititiko", 1.5F));
        add(ticketStudent);
        // students should complete this with the rest types of tickets
    }

    // creates the rest buttons (cancel, finish, display area)
    void createOtherButtons() {
        // cancelButton = new JButton("Cancel") {
        cancelButton = new JButton("Cancel");
        cancelButton.setBackground(Color.RED);
        cancelButton.setOpaque(true);
        cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent e) {
                System.out.println("Cancel");
                APPLICATION.bts.cancel(); // calls the corresponding method of the model
                APPLICATION.updateDisplay();
            }
        });
    }
}

```



```

    }
    });
    add(cancelButton);

    finishButton = new JButton("Finish");
    finishButton.setBackground(Color.GREEN);
    finishButton.setOpaque(true);
    finishButton.addActionListener(new ActionListener() {
        public void actionPerformed(java.awt.event.ActionEvent e) {
            System.out.println("Finish");
            APPLICATION.bts.finish(); // calls the corresponding method of the model
            APPLICATION.updateDisplay();
        }
    });
    add(finishButton);

    displayArea = new JLabel("Anoixame kai sas perimenoyme!!!");
    add(displayArea);
}

static void updateDisplay() {
    displayArea.setText("Katastasi: " + bts.getFullState());
}

APPLICATION() {
    // CORE PART
    bts = new BusTicketsSeller(10); // 10 = stock of tickets

    // GUI PART: size, layout, title
    setBounds(100, 100, 900, 200); //x, y, width, height
    setLayout(new GridLayout(4, 12)); // rows, columns
    setVisible(true);
    setTitle(appName);

    // GUI PART: calling the methods that create buttons
    createCoinButtons();
    createTicketButtons();
    createOtherButtons();
    setVisible(true);
    // that's all
}

public static void main(String[] args) {
    System.out.println(appName);
    APPLICATION app = new APPLICATION();
}
}

```

ΥΣΤΕΡΟΓΡΑΦΟ

Καλό είναι να μελετάτε παράλληλα και τα παλαιότερα θέματα προόδου και για να κατανοήσετε τους εκπαιδευτικούς στόχους, και για να βοηθηθείτε στην Α2 και ως προετοιμασία για την επικείμενη εξέταση προόδου.

Καλή εργασία!

Τέλος Α2

