Opdrachten INFDEV21-3

De onderstaande opdrachten dienen in Java of C# gemaakt te worden.

# Opdracht 1

Schrijf een programma dat eenmalig een getal genereert tussen de 0 en de 100, en vervolgens de gebruiker vraagt dit getal te gokken. De gebruiker heeft maximaal 7 pogingen.

Indien de gebruiker het juiste getal raadt waarbij het aantal pogingen binnen het maximale aantal pogingen is, dan krijgt de gebruiker te zien: “Je hebt juist gegokt”, en anders “Helaas niet gelukt”.

Indien bij een poging de gebruiker te laag gokt, moet worden uitgeprint: te laag. Indien te hoog gegokt, dan “te hoog”. Indien juist gegokt, dan “Je hebt juist gegokt” en moet het programma **direct afbreken**.

Het vragen naar user input kan in Java d.m.v. javax.swing.Joptionpane.showInputDialog of via de iets object georienteerde manier: new Scanner(System.in).next… (Google hier naar).

Omzetten van een String naar een getal gaat in Java met Integer.parseInt(deTekst…).

# Opdracht 2

print "H H I"

print "H H"

print "H H I"

print "HHHHH I"

print "H H I"

print "H H I"

print "H H I"

print 1 + 1

print "H H I"

print "H H"

print "H H I"

print "HHHHH I"

print "H H I"

print "H H I"

print "H H I"

De bovenstaande Python code bevat duplicated code. Herschrijf de bovenstaande code (in Java of C#), pas een functie toe om de duplicated code verwijderen.

(Gebruik “static” functies, zodat je ze meteen kan aanroepen.)

# Opdracht 3

Vraag de gebruiker 10 keer achter elkaar een zin/woord(en) in te voeren. Elke keer dat de gebruiker iets invoert (en op enter gedrukt) sla je datgene op in de array. (Let op, gebruik een array, en niet een arraylist.)

Bereken vervolgens het totaal aantal ingevoerde tekens (letters, nummers, spaties etc), en print deze hoeveelheid uit.

In Java kan je de lengte van een string krijgen met: mijnStringVariabele.length().

# Opdracht 4Afbeeldingsresultaat voor wat is een rechthoek

Rechthoeken, hiernaast zie je er een paar. Een rechthoek bestaat uit een hoogte en een breedte. Met een gegeven hoogte en breedte, kan je dus een rechthoek weergeven.  
In een applicatie die ontwikkeld moet worden, moeten rechthoeken gebruikt worden. Alles dat kwa figuur complexer is dan een rechthoek, wordt in de applicatie een ComplexFiguur genoemd. Dit is bijvoorbeeld het geval indien er meer dan 4 hoeken aanwezig zijn in het figuur.

Opdracht:

* Schrijf een klas Rechthoek.
* Zorg ervoor dat de constructor van Rechthoek de eigenschappen/attributes *hoogte* en *breedte* instelt, die als parameters worden meegegeven.
* Schrijf een klas ComplexFiguur. (Deze klas heeft verder geen eigenschappen/state en geen methodes/functies.)
* Maak **in de Rechthoek klas** een functie/methode genaamd *ernaast()*. De methode *ernaast()* accepteert een ander rechthoek object als parameter, en produceert een nieuwe rechthoek als return value. De methode ernaast() plaats twee rechthoeken horizontaal tegen/naast elkaar indien ze dezelfde hoogte hebben. Er ontstaat dan een nieuwe rechthoek/object waarvan de breedte dus de som van de breedtes van de twee naast elkaar geplaatste rechthoeken is, en de hoogte gelijk gebleven is. Deze nieuwe rechthoek moet als return value worden teruggegeven. De bestaande rechthoeken mogen niet worden gewijzigd.  
  Indien de hoogte echter afwijkt, ontstaat er geen rechthoek, maar een complex figuur (want er zijn dan meer dan 4 hoeken). Geef dan een ComplexFiguur object als return value terug.
* Schrijf clientcode waarin je gebruik maakt van de Rechthoek klas:
  + Maak twee rechthoek objecten met dezelfde grootte.
  + Roep op de eerste rechthoek de ernaast() methode aan en geef de tweede rechthoek mee als argument/paramter.
  + Print de breedte van de daaruit voortkomende rechthoek uit.

# Opdracht 5

Maken, alle opdrachten van <http://med.hro.nl/oelew/tinpro01-2/opd1.pdf>  
(Dit document komt van voltijd Technische Informatica.)

Met een “pointer” wordt een referentie naar een object bedoeld. In opdracht 4 (van die pdf) dient de method er dus als volgt uit te zien: public static Auto snelsteAuto(Auto[] autos) {...}

Met “modifiers” wordt de visibility bedoeld (bijv. public, private, protected, package-private).

# Opdracht 6

Maken, alle opdrachten van <http://med.hro.nl/oelew/tinpro01-2/opd2.pdf>

# Opdracht 7

Maken, alle opdrachten van <http://med.hro.nl/oelew/tinpro01-2/opd3.pdf>

# Opdracht 8

Gegeven de onderstaande code. Pas overerving (inheritance) op dusdanige wijze toe dat code duplicatie wordt geminimaliseerd, en eventuele onnodige expliciete casts en instanceof operators verdwijnen.

**public class** App {  
 **public static void** main(String[] args) {  
 Administratie bedrijfsbureau = **new** Administratie();  
  
 Docent james = **new** Docent();  
 james.setVoornaam(**"James"**);  
 james.setAchternaam(**"Gosling"**);  
 james.setId(**"gosja"**);  
  
 Docent guido = **new** Docent();  
 guido.setVoornaam(**"Guido"**);  
 guido.setTussenvoegsel(**"van"**);  
 guido.setAchternaam(**"Rossum"**);  
 guido.setId(**"guiro"**);  
  
 bedrijfsbureau.registreer(james);  
 bedrijfsbureau.registreer(guido);  
  
 Student foo = **new** Student();  
 foo.setVoornaam(**"Foo"**);  
 foo.setAchternaam(**"Bar"**);  
 foo.setMentor(james);  
 foo.setId(**"23456789"**);  
  
 Student hello = **new** Student();  
 hello.setVoornaam(**"Hello"**);  
 hello.setAchternaam(**"World"**);  
 hello.setMentor(guido);  
 hello.setId(**"12345678"**);  
  
 bedrijfsbureau.registreer(foo);  
 bedrijfsbureau.registreer(hello);  
  
 bedrijfsbureau.printAllePersonen();  
 }  
}  
  
**class** Administratie {  
 **private** Object[] **personen** = **new** Object[100]; *// De school heeft een max capaciteit van 100 docenten/studenten* **private int aantalPersonen** = 0;  
  
 **public void** registreer(Student student) {  
 **personen**[**aantalPersonen**] = student;  
 **aantalPersonen**++;  
 }  
  
 **public void** registreer(Docent docent) {  
 **personen**[**aantalPersonen**] = docent;  
 **aantalPersonen**++;  
 }  
  
 **public void** printAllePersonen() {  
 **for** (**int** i = 0; i < **aantalPersonen**; i++) {  
 Object persoon = **personen**[i];  
  
 **if** (persoon **instanceof** Docent) {  
 Docent docent = (Docent) persoon;  
 System.***out***.println(docent.getVolledigeNaam());  
 } **else if** (persoon **instanceof** Student) {  
 Student student = (Student) persoon;  
 System.***out***.println(student.getVolledigeNaam());  
 } **else** {  
 System.***out***.println(**"Error! "** + **" het object "** + persoon + **" lijkt geen persoon/docent/student te zijn!"**);  
 }  
 }  
 }  
}  
  
**class** Docent {  
 **private** String **id**; *// studentnr* **private** String **voornaam**;  
 **private** String **achternaam**;  
 **private** String **tussenvoegsel**;  
 **private** String **afdeling**;  
 **private** Adres **woonAdres**;  
  
 **public** String getId() {  
 **return id**;  
 }  
  
 **public void** setId(String id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getVoornaam() {  
 **return voornaam**;  
 }  
  
 **public void** setVoornaam(String voornaam) {  
 **this**.**voornaam** = voornaam;  
 }  
  
 **public** String getAchternaam() {  
 **return achternaam**;  
 }  
  
 **public void** setAchternaam(String achternaam) {  
 **this**.**achternaam** = achternaam;  
 }  
  
 **public** String getTussenvoegsel() {  
 **return tussenvoegsel**;  
 }  
  
 **public void** setTussenvoegsel(String tussenvoegsel) {  
 **this**.**tussenvoegsel** = tussenvoegsel;  
 }  
  
 **public** String getVolledigeNaam() {  
 **if** (**tussenvoegsel** == **null**) {  
 **return voornaam** + **" "** + **achternaam**;  
 } **else** {  
 **return voornaam** + **" "** + **tussenvoegsel** + **" "** + **achternaam**;  
 }  
 }  
  
 **public** String getAfdeling() {  
 **return afdeling**;  
 }  
  
 **public void** setAfdeling(String afdeling) {  
 **this**.**afdeling** = afdeling;  
 }  
  
 **public** Adres getWoonAdres() {  
 **return woonAdres**;  
 }  
  
 **public void** setWoonAdres(Adres woonAdres) {  
 **this**.**woonAdres** = woonAdres;  
 }  
}  
  
**class** Student {  
 **private** String **id**; *// studentnr* **private** String **voornaam**;  
 **private** String **achternaam**;  
 **private** String **tussenvoegsel**;  
 **private** String **klas**;  
 **private** Adres **woonAdres**;  
 **private** Adres **werkAdres**;  
 **private** Docent **mentor**;  
  
 **public** String getId() {  
 **return id**;  
 }  
  
 **public void** setId(String id) {  
 **this**.**id** = id;  
 }  
  
 **public** String getVoornaam() {  
 **return voornaam**;  
 }  
  
 **public void** setVoornaam(String voornaam) {  
 **this**.**voornaam** = voornaam;  
 }  
  
 **public** String getAchternaam() {  
 **return achternaam**;  
 }  
  
 **public void** setAchternaam(String achternaam) {  
 **this**.**achternaam** = achternaam;  
 }  
  
 **public** String getTussenvoegsel() {  
 **return tussenvoegsel**;  
 }  
  
 **public void** setTussenvoegsel(String tussenvoegsel) {  
 **this**.**tussenvoegsel** = tussenvoegsel;  
 }  
  
 **public** String getVolledigeNaam() {  
 **if** (**tussenvoegsel** == **null**) {  
 **return voornaam** + **" "** + **achternaam**;  
 } **else** {  
 **return voornaam** + **" "** + **tussenvoegsel** + **" "** + **achternaam**;  
 }  
 }  
  
 **public** String getKlas() {  
 **return klas**;  
 }  
  
 **public void** setKlas(String klas) {  
 **this**.**klas** = klas;  
 }  
  
 **public** Adres getWoonAdres() {  
 **return woonAdres**;  
 }  
  
 **public void** setWoonAdres(Adres woonAdres) {  
 **this**.**woonAdres** = woonAdres;  
 }  
  
 **public** Adres getWerkAdres() {  
 **return werkAdres**;  
 }  
  
 **public void** setWerkAdres(Adres werkAdres) {  
 **this**.**werkAdres** = werkAdres;  
 }  
  
 **public** Docent getMentor() {  
 **return mentor**;  
 }  
  
 **public void** setMentor(Docent mentor) {  
 **this**.**mentor** = mentor;  
 }  
}  
  
**class** Adres {  
 **private** String **straat**;  
 **private** String **huisnummer**;  
 **private** String **postcode**;  
 **private** String **plaats**;  
  
 **public** Adres(String straat, String huisnummer, String postcode, String plaats) {  
 **this**.**straat** = straat;  
 **this**.**huisnummer** = huisnummer;  
 **this**.**postcode** = postcode;  
 **this**.**plaats** = plaats;  
 }  
  
 **public** String getStraat() {  
 **return straat**;  
 }  
  
 **public** String getHuisnummer() {  
 **return huisnummer**;  
 }  
  
 **public** String getPostcode() {  
 **return postcode**;  
 }  
  
 **public** String getPlaats() {  
 **return plaats**;  
 }  
}

# Opdracht 9

Maken alle opdrachten van <http://med.hro.nl/oelew/tinpro01-2/opd4.pdf>

# Opdracht 10

**class** Cat {  
 **public void** produceSound() {  
 System. ***out*** .println( **"mmiiiaauw...."** );  
 }  
 **public void** eat() {  
 System. ***out*** .println( **"Eating a bird...."** );  
 }  
}  
**class** Speaker {  
 **public void** powerOn() {  
 System. ***out*** .println( **"Speakers are on..."** );  
 }  
 **public void** produceSound() {  
 System. ***out*** .println( **"Making music"** );  
 }  
}  
**class** SoundTester {  
 **private** Object[] **soundProducers** = **new** Object[100];  
 **private int counter** = 0;  
  
 **public void** addSoundObject(Object o) {  
 **if** (o **instanceof** Speaker || o **instanceof** Cat) {  
 **soundProducers**[**counter**] = o;  
 **counter**++;  
 }  
 }  
 **public void** playSoundProducers() {  
 **for** (Object soundProducer : **soundProducers** ) {  
 **if** (soundProducer **instanceof** Cat) {  
 Cat c = (Cat) soundProducer;  
 c.produceSound();  
 } **else** {  
 Speaker s = (Speaker) soundProducer;  
 s.produceSound();  
 }  
 }  
 }  
}  
**public class** Main {  
 **public static void** main(String[] args) {  
 SoundTester tester = **new** SoundTester();  
 tester.addSoundObject( **new** Cat());  
 tester.addSoundObject( **new** Speaker());  
 tester.playSoundProducers();  
 }  
}

De bovenstaande (werkende) code bestaat uit een Cat en een Speaker class. Beiden kunnen geluid

produceren. Aan een SoundTester object kunnen geluidproducerende objecten (zoals een Cat en een Speaker) worden toegevoegd, waarna met de playSoundProducers() method de geluidproducerende objecten getest worden door ze geluid te laten produceren.

Het probleem van de code is dat deze niet netjes is: er is geen polymorphisme toegepast, waardoor

de code complexer is dan nodig. Het ontbreken van polymorphisme is onder andere te herkennen aan het gebruik van zeer algemene types, zoals Object, en het gebruik van instanceof. Dit laatste

resulteert in het gebruik van if-statements, waardoor de complexiteit onnodig wordt vergroot.

Opdracht: pas polymorphisme op dusdanige wijze toe dat de **if-statements, instanceof-operators en expliciete casts verdwijnen** . Het is **niet toegestaan om nieuwe classen te introduceren of superclass relaties aan te maken** . Het introduceren van interfaces is wel toegestaan.

# Opdracht 11

Maken: <https://www.dropbox.com/s/i5cu4n1l1gxxkyz/Opdracht4.docx?dl=0>

# Opdracht 12 (Optioneel)

Maken oefententamens:

1. <https://www.dropbox.com/s/24d67k0zdn04sb7/INFDEV02_3_sample_exam1.pdf?dl=0>
2. <https://www.dropbox.com/s/yswjh6j66eg05pl/INFDEV02_3_sample_exam2.pdf?dl=0>
3. <https://www.dropbox.com/s/c2dyfy3cprjzsa4/INFDEV02_3_sample_exam3.pdf?dl=0>

# Opdracht 13

Bekijk <https://www.youtube.com/watch?v=MGEx35FjBuo> over de State Design Pattern.

In de video gebruiken ze de State Design Pattern om een pinautomaat (ATM=Automatic Telling Machine) te modeleren/ontwerpen in code. Een design pattern is een bepaalde structuur van hoe classes ontworpen zijn, welke relaties bepaalde classes met elkaar hebben etc. Design patterns kunnen leiden tot nettere object georienteerde code, door bijvoorbeeld complexe code (bijv. if statements) te verminderen, maar daar tegenover staat dat er meestal wel meer classes gemaakt moeten worden. (Niet dat dat slecht is, maar als je het design pattern niet kent, kan de structuur tussen de classen wat onduidelijk zijn.)

Indien er geen State Design Pattern gebruikt zou worden in deze case, dan zou je waarschijnlijk methods krijgen met daarin relatief veel if-statements, waarbij de if-statements verspreid over de methods ongeveer steeds dezelfde structuur hebben, wat een vorm is van duplicated code, en dus ongewenst is.

In het voorbeeld is er een onderwerp/context: de pinautomaat (ATMMachine). De ATMMachine heeft verschillende toestanden: HasCard, NoCard, HasPin en NoCash. Er is altijd maar één toestand tegelijkertijd actief/ingesteld op de ATMMachine. Als je als klant bij een pinautomaat komt, dan is de toestand: NoCard, want je hebt nog niks gedaan. Zodra je je bankpas erin steekt, gaat de toestand NoCard via de transitie/actie insertCard() over naar de toestand HasCard.  
Naar insertCard() zijn er nog andere transities/acties/handelingen: ejectCard, insertPint(int) en requestCash(int).

Het gebruik van de State Design Pattern zorgt er voor dat de focus op de transities (methods) verschuift naar een focus op de toestanden (gemodeleerd als classen).

Opdracht: ook al begrijp je het idee van de State Design Pattern niet helemaal, probeer het over te nemen voor de volgende case, en programmeer de bijbehorende code. Een lift in een gebouw kan zich in de volgende toestanden bevinden: 1) de deur open, 2) deur gesloten, 3) in beweging naar een verdieping. Naast de toestanden zijn er de volgende transitities/acties: verdieping instellen, deur openen, deur sluiten, lift bewegen.



|  |  |  |  |
| --- | --- | --- | --- |
|  | **Toestanden** | | |
|  | Deur open | Deur gesloten | In beweging |
| Stel verdieping in | niet toegestaan | Stel de verdieping in. En ga naar In beweging toestand. | niet toegestaan |
| Beweeg lift | niet toegestaan | niet toegestaan | Beweeg de lift naar de juiste verdieping. Indien bereikt, ga naar Deur gesloten toestand. |
| Open deur | geen effect | Verander naar Deur open toestand. | niet toegestaan |
| Sluit deur | Verander naar Deur gesloten toestand | geen effect | niet toegestaan |