LULEÅ TEKNISKA UNIVERSITET

Tentamen i
Objektorienterad programmering och design
Totala antalet uppgifter: 5
Lärare: Håkan Jonsson, 491000, 073-820 1700
Resultatet offentliggörs senast: 2015-04-10.

Kurskod	D0010E
Datum	2015-03-20
Skrivtid	5 tim

Tillåtna hjälpmedel: Inga.

OBS! Lösningar får *inte* baseras på fördefinierade klasser ur t ex Javas standardbibliotek annat än där detta uttryckligen tillåts. Sådana lösningar ger inga poäng.

---

1. Teori

a) Klassen `X` i paketet `x` ärver den publika klassen `Y` i paketet `y`. I `x` finns även klassen `A`, och i `y` dessutom `B`.

1. Rita UML-diagrammet. (1p)
2. Antag att `Y` innehåller en metod `m` deklarerad `protected`. I vilken/vilka av klasserna `A`, `B`, `X` och `Y` kan den då *inte* (OBS! *inte*) användas? (1p)

b) Ange något bra och något dåligt med variabler deklarerade `static`. (2p)

c) Vad innebär *arv*? (1p)

d) Vad betyder det att en metod är *polymorf*? (1p)

2. Svansar och delar

I dessa deluppgifter får du använda strängmetoderna `public char charAt(int i)` som ger tecknet på index `i` (som börjar på 0) och `public int length()` som ger strängars längd.

a) Skriv en iterativ metod `svansen` som givet en sträng s med längden ℓ och ett positivt heltal k, sådant att $1 \leq k \leq \ell$, ger tillbaka en sträng med de $(\ell - k) + 1$ sista tecknen ur s. Anropet `svansen("abcdefgh", 4)` ska t ex returnera `"defgh"`.

Om $k > \ell$ returneras tomma strängen. Om $k < 1$ ska däremot istället undantaget `IndexOutOfBoundsException` kastas. (3p)

b) En sträng s är en delsträng i en annan sträng s' om antingen s är tom eller tecknen i s även förekommer i s' och i ordning. Strängen `"nöd"` är delsträng i såväl `"nödanrop"`, `"andnöd"`, `"nörd"` och `"anförde"` men varken `"nöta"` (tecknet `d` saknas) eller `"drönare"` (alla tecknen finns med men ordningen är fel).

Skriv en rekursiv metod `isSubString` som avgör om en sträng är en delsträng. Du får använda `svansen` ur a) även om du inte löst den deluppgiften. (3p)

Ledning: s är en delsträng om den är tom men inte om den är icke-tom samtidigt som s' är tom. Annars beror det på de första tecknen is s och s': Är de lika så är s delsträng i s' om `svansen(`s`,2)` är delsträng i `svansen(`s'`,2)`. Är de olika så är s delsträng i s' om s är delsträng i `svansen(`s'`,2)`.

3. Ett kort kortproblem

Skriv en klass för att hålla reda på en mängd spelkort (en *hand*) valda ur en kortlek med 52 kort.

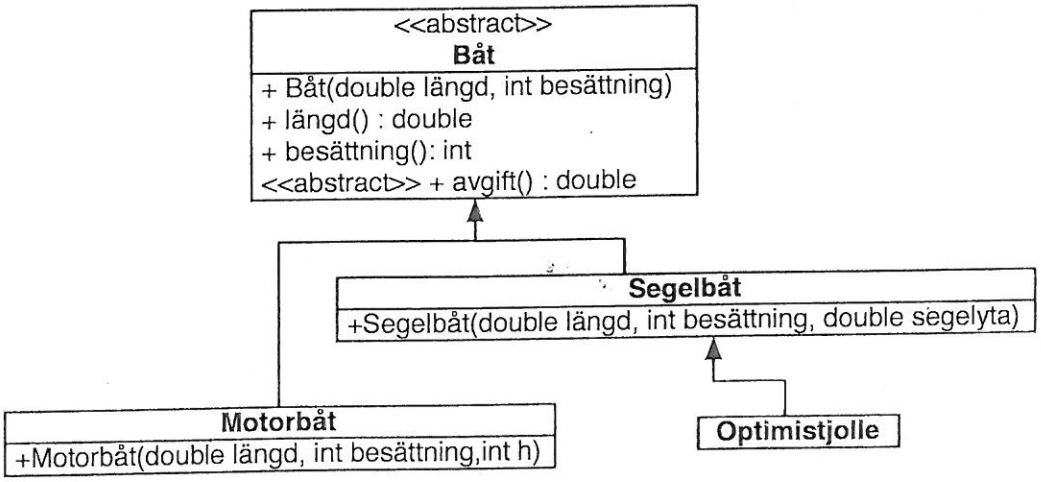
Med `public void insert(Card c, int k)` stoppar man in kort `c` på position `k`. Då flyttas varje kort på position $i > k$ till position $i+1$ (för att skapa utrymme för det insatta kortet). Har handen n kort kan insättning endast ske på positionerna 1 till $n+1$. Med `public Card take(int k)` tar man kortet på position `k` från handen. Då flyttas alla kort på position $i > k$ till position $i-1$. Metoden `public int size()` ger slutligen antalet kort handen innehåller.

Figur 1: Kortet ♣7 sätts in på position 4. Kortet ♣10 flyttas då från position 4 till 5.

Du får utgå från att spelkort är objekt av den redan skrivna klassen `Card`[1] samt att alla kort som sätts in är unika. Metoderna ska kasta undantag om de för övrigt används felaktigt. (6p)

Ledning: Hur många kort kan det som mest finnas i en hand?

4. Båtar båtar?



Figur 2: UML-diagram.

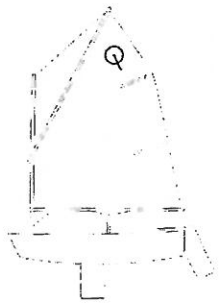
Implementera klasserna i figur 2 för att representera båtar. OBS! För att lösa uppgiften kan du behöva göra tillägg av variabler, metoder, klasser mm utöver vad som anges i UML-diagrammet. Undantaget `IllegalBoat` ska kastas när så är lämpligt.

En *Båt* definieras av dess längd i meter och storleken på besättningen. Metoderna `längd` och `besättning` ger tillbaka båtens längd respektive antalet i besättningen. Den abstrakta metoden *avgift* står för en avgift båtägare måste betala årligen.

En *Motorbåt* är en båt med en motor som har ett antalet hästkrafter h. För motorbåtar är avgiften $A(h) = h^2$.

En *Segelbåt* är en båt med en längd som inte får överstiga 12 meter samt en segelyta s. Avgiften för en segelbåt är $A(s) = 0.6s + 20$.

En *Optimistjolle* är en segelbåt som är 2.3m lång och endast har en person i besättningen. Segelytan är för en optimistjolle 3.5m^2 och avgiften är endast 75% av vad avgiften skulle vara om man vid beräkningen betraktade den endast som en segelbåt istället för en optimistjolle.



Figur 3: Optimistjolle.

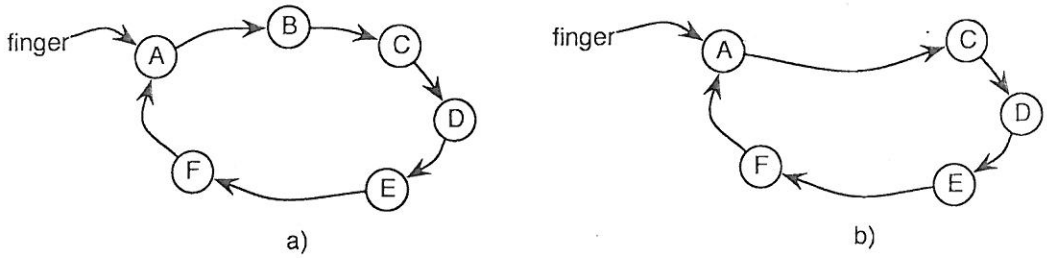
(6p)

[1] Du ska alltså inte implementera `Card`.

5. Cirkulär lista

Skriv en klass `CircularList` för cirkulära listor med interna (enkellänkade) listnoder.

En lista har ett (1) *finger* som markerar ett av elementen, då listan innehåller sådana (figur 4a), eller som är odefinierat då listan är tom t ex då den är alldeles nyskapad. Listor



Figur 4: a) En lista med elementen A, B, C, D, E och F i vilken fingret pekar på A. b) Resultatet av `remove` då fingret står på A.

har fem metoder:

public void forward() Flyttar fingret till nästa element. I figur 4a) skulle fingret då komma att peka på B.

Om listan innehåller endast ett element pekar fingret fortfarande på detta efter anrop till denna metod. Undantaget `IllegalStateException` ska kastas om listan är tom.

public void remove() Tar bort elementet efter det som fingret pekar på. Ett exempel visas i figur 4. Fingret pekar först på A som i figur 4a. Då `remove` anropas tas B bort.

`remove` på en lista med endast ett element ska ta bort det enda elementet (och göra listan tom) medan undantaget `NoSuchElementException` ska kastas om `remove` anropas och listan redan är tom.

public Object item() Returnerar en referens till det element som fingret pekar på. Undantaget `NoSuchElementException` ska kastas om listan är tom.

public Object nextItem() Returnerar en referens till elementet *efter* det som fingret pekar på, dvs det element som ett anrop till `remove` skulle ta bort. I figur 4a) pekar fingret på A och `nextItem` skulle returnera B.

Finns det endast ett element i listan så ger `nextItem` samma resultat som `item`. Undantaget `NoSuchElementException` ska kastas om listan är tom.

public void insert(Object item) Sätter in `item` direkt efter det element fingret pekar på. Om vi i figur 4b sätter in B så blir listan den i figur 4a.

Efter insättning i en tom lista ska fingret peka på det insatta elementet. Detta enda element "följer efter sig självt" i listan (se beskrivningen av `forward`).

Andra vanligt förekommande (och vettiga) metoder, som att t ex för att få en listas storlek eller kunna gå runt åt andra hållet ("backward"), behöver inte implementeras. (6p)