Diskussionsfråga 1

Föreläsningen	***Stack***	***Anmärkning***
Skapa stack		Sker i konstruktorn
Förstöra stack	clear	Den tömmer bara, GC ”förstör” den
Stacken tom?	isEmpty	
Stacken full?		Är den full har vi andra problem (minne)
Stackens storlek?		FINNS EJ!!! Borde finnas! ☹
Push	enqueue	
Pop	dequeue	
Peek	getFront	

Diskussionsfråga 2

Stack	***Java.util.Stack***	***Anmärkning***
clear	clear	
isEmpty	empty	
enqueue	push	
dequeue	pop	
getFront	peek	
	search	!
	extends Vector	Dvs massa mer metoder (PS. även andra interface)

Diskussionsfråga 3

Den ärver av vektor, vilket gör att man kan ta sig runt ”stack” delen och använda den som en lista (Vector).

Diskussionsfråga 4

Man kan returnera ”null” (inget) istället för att kasta fel när man försöker hämta något ur listan.

- Riskerar lättare att få null exceptions
- Kräver mer jobb

Man kan deklarera metoderna som kastande(”… throws …”)

- Kräver felhantering (större sannolikhet att felen kan bli åtgärdade)
- Klumpig
- Kan inducera felaktigt beteende (”catch” istället för ”if”, döljande av symtom istället för lösning av problem)

Tysta ”throws” (runtime exceptions)

- Informativt vid felsökning

Return-null:

```
Data data = null;
while((data = stack.pop()) != null)
{
    proccess(data);
}
```

Not-return-null:

```
while(!stack.empty())
{
    proccess(stack.pop());
}
```

Diskussionsfråga 5

Med ”runtime exceptions”

Diskussionsfråga 6

```java
public interface MyList<T>
{
    public int getSize();

    public boolean isEmpty();

    @SuppressWarnings("unchecked")
    public MyList<T> add(T... element);

    @SuppressWarnings("unchecked")
    public MyList<T> add(int start, T... element);

    public boolean contains(T object);

    public int indexOf(T object);

    @SuppressWarnings("unchecked")
    public int remove(T... element);

    public T removeAt(int index);

    public T get(int index);

    public MyList<T> clear();
}
```