Iteration Exercise(s)

Extend the standard Java Vector class to create a new *MyVector* class, which _also_ has an internal iterator.

Name the internal iterator method *doAll(…),* and it should take a single argument which is the functor that encapsulated the processing for each element in the loop. Use our standard Functor interface, which includes two methods;

- *Functor<T>.compute(T element)*
- *Functor<T>.getValue( )*

Do this in several steps;

	Internal/External	**Complete/selective**	
	External	Complete	✓ (Standard Java)
1.	Internal	Complete	w/ *sum* functor
1.b	Internal	Complete	w/ *average* functor
2.	External	Selective	(repeat last two, w/ selection)
3.	Internal	Selective	[Optional]
4.	External	Selective + *Iterable*	[Optional] ≈ *view*

For summation example;
create a collection with *Integers* { 1, 2, 3, … 10}, and find the sum of all the values.

For selection, use a *Predicate* with criteria: { x>2 }, and find the sum of all the numbers that meet this criteria.

Make your *SumFunctor* class concrete (not generic), by having it extend the concrete class *Functor<Integer>*.
[You can ignore generics if you find that they confuse your design for a first version.]

To keep the *Functor* class simple, you do not have to make Generic over return type <R>, but if possible this is a good addition.

To test your selective$|_2$ function, find the sum of all numbers in the list { 1..10 } which are >2.

For (External Selective):
For #3 you need to build a new iterator method which returns an *Iterator* object, but it needs ot have a different signature: *Iterator iterator(Predicate),* since it needs to be created using this new information on what selective values to return.

The object it returns must meet the standard *Iterator* interface, but its methods will be basically a filter; it will internally use the default standard *iterator()* objects form the Vector class, but will then also use the *Predicate* given to determine which of these (complete set of) elements should be returned to the user.