1. D -c
2. E -w Just ChessPiece would do fine as a abstraction.
3. A -X is a valid answer
4. D -w B is the correct answer. Type checking occurs at the runtime only as it may not be available at compilation. Type erasure only at compilation. Type Inference as well.
5. E -w A, typecasting happens at compile time, but is checked during runtime. Accessibility checks happen both during run and compile time
6. C -w E java generics are invariant
7. C -w E
8. B -c
9. A -w Members of a stream should be final or effectively final.
10. A -w Read CAREFULLY
11. A - c
12. D
13. 1. Return type of curry should be a value of type R
    2. Return new Function <S,Function<T,Function<U,R>>>{

@Override

Public Function<T,Function<U,R>> apply(S val){

Return new Function

1. Return new Predicate<>(x->p.test(f.apply(x))); -C
2. Static<T> Boolean isSubStream(Stream<T> s, Stream<T> t){

Return s.allMatch(x->t.anyMatch(y-> y.equals(x)));

}

Static<T> Boolean isSubStream(Stream<? extends T> s, Stream<T> t)

* 1. Undoable<Integer> length(String str){

Deque<Object> d = new LinkedList<>(str); **CANNOT DIRECTLY ADD**

Return new Undoable<>(str.size(),d);

}

* 1. Public <R>Undoable<R> flatMap(Function<T,Undoable<R>> mapper){

Undoable<R> newVal = mapper.apply(value);

Deque<Object> d = new LinkedList<>(history);

d.add(value);

Return new Undoable<R>(newVal.value,d);

}

* 1. The compiler cannot be sure as to what type from the generic typing, “R”, the object that is retrieved from the list will be, as the history is stored as a Deque of objects, due to type erasure. But since R is erased during compile time, the runtime JVM cannot safely check the typing to ensure that it matches.
  2. The
  3. Throw new CannotUndoException();

1. 1. Fgce fgce

**IDENTITY is first evaluated in the stream then**

* 1. Public static <T>LazyList<T> Concat(LazyList<T> l1,LazyList<T> l2){

If (l1.isEmpty()){

Return l2;

}else{

Return new LazyList<T>(()->l1.head.get(),()->concat(l1.tail.get(),l2));

}

}