Actividad de lectura:

1. Te permite definir polimorfismo entre clases que no comparten una herencia. Esto hace que los sistemas puedan crecer de mejor manera, ya que pueden aparecer nuevas clases que no necesariamente deban ser subclases de una clase existente pero que sí sean polimórficas con el resto.
2. Porque no necesariamente las clases polimórficas que quiero definir deberían ser subclases de una clase abstracta. Quizás solo comparten un par de mensajes con la clase abstracta, pero eso no la hace subclase necesariamente, pero si las hace polimórficas.
3. Las clases abstractas permiten establecer relaciones más directas entre clases, es decir, que una subclase es su superclase pero con alguna característica particular. Además, cuando una clase es subclase de una clase abstracta, entonces no necesito definir muchos de los mensajes que heredé de mi superclase y solo debo implementar los mensajes abstractos. También tengo la ventaja de heredar las variables de instancia. En las interfaces, en cambio, solo puedo listar los mensajes y las variables de instancia que quiero que compartan todas las clases que se implementen de mi. Esto significa que en cada clase que se implemente de mi interfaz voy a tener que implementar los métodos de los mensajes de la interfaz.
4. No, una interface se usa para definir polimorfismo entre clases, y se pueden instanciar clases definidas por una interface, pero una interface no se puede instanciar. Esto tiene sentido ya que una interface no define sus métodos, sólo define qué mensajes tienen que poder responder las clases que se implementen de ella.
5. Porque si quiero incrementarlo, entonces todas las clases que se implementen de esa interface también van a tener que incrementar sus protocolos, y eso puede traer problemas. Por eso es bueno tener muchas interfaces específicas y no muchas generales.
6. Porque Smalltalk es un lenguaje con tipado dinámico. Esto significa que no va a chequear tipos antes de compilar. Entonces podes tener clases polimórficas entre sí y agruparlas sin problemas, siempre y cuando te asegures que esas clases sean efectivamente polimórficas para poder pasarle el mismo mensaje a dos elementos del grupo y que lo conozcan ambos. Esto en Java no se puede ya que al ser tipado estático, las listas, por ejemplo, tienen que tener un tipo específico. Como solo le puedo poner un tipo, si quiero expandir mi sistema y tener una lista de clases polimórficas pero que no comparten herencia, debo implementar una interfaz que asegure que mis clases son polimórficas y poner esta interfaz como tipo de la lista.

Interfaces, colecciones y otras yerbas:

Para XXX: podriamos darle cualquier clase que esté implementada con la interfaz List (ArrayList y LinkedList en este caso), ya que estas van a poder

responder el mensaje get(). El mensaje getFirstFrom() te devolverá el primer elemento de la lista. Esta lista siempre va a ser ordenada.

Para YYY: podríamos darle cualquier clase que esté implementada con la interfaz Collection, ya que van a poder responder el mensaje add() porque así lo define la interfaz. El mensaje addLast() va a agregar un elemento al final de la colección.

Para ZZZ: En este caso falta definir x e y, pero si x e y fueran from y to respectivamente, podríamos darle cualquier clase que esté implementada con la interfaz List. El mensaje getSubCollection() nos devolverá una colección formada por los elemento desde el índice from inclusive hasta el índice to.

Para WWW: podríamos darle cualquier clase que esté implementada con la interfaz Collection, ya que va a poder responder el mensaje contains(). El mensaje containsElement() nos devuelve un booleano que representa si un valor esta presente en la colección o no.