|  |
| --- |
| **Exercise 12.2**  After creating a comment on the MessagePost object it is also displayed when listing the news feed, even though the MessagePost was added to the news feed before setting the comment.  This is because the NewsFeed has a reference to the MessagePost object and uses this reference to get the information about the object each time it prints the list. Only one MessagePost object has been created so the NewsFeed must be storing the one that has the comment. |
| **Exercise 12.4**   When the 'extends Item' is removed from the source of the MessagePost class, then the inheritance arrow in the diagram disappears. |
| **Exercise 12.5**  Yes, it is possible to call inherited methods through the sub-menu called '*inherited from Post*' |
| **Exercise 12.6**  **Add the** getUserName **method to** Post**:**  public String getUserName()  {  return username;  }  Define printShortSummary in MessagePost:  public void printShortSummary()  {  System.out.println("Message post from " + getUserName());  } |
| **Exercise 12.7**  First, the *Step Into* button takes us to the superclass constructor, which on the subsequent *Step Into* initializes the fields: username, timestamp, likes and comments. Then it returns back to the MessagePost constructor and initializes the last field: message. |
| **Exercise 12.8**  The EventPost:  /\*\*  \* This class stores information about a post in a social network news feed.  \* The main part of the post consists of a (possibly multi-line)  \* text message. Other data, such as author and time, are also stored.  \*  \* @author Michael Kölling and David J. Barnes  \* @version 0.2  \*/  public class EventPost extends Post  {  // The type of event.  private String eventType;  /\*\*  \* Constructor for objects of class EventPost  \* @param author The author of the post.  \* @param eventType The type of event.  \*/  public EventPost(String author, String eventType)  {  super(author);  this.eventType = eventType;  }  /\*\*  \* Return the type of event.  \*  \* @return The type of event.  \*/  public String getEventType()  {  return eventType;  }  } |
| **Exercise 12.9**.  One possible hierarchy: |
| **Exercise 12.10**  A touch pad and mouse are both input devices for a computer. They are very similar and they could either have a common superclass (InputDevice) or one could be a superclass of the other. |
| **Exercise 12.11**  Argument for a square being a subclass of a rectangle:  - a square is just a rectangle where the sides are restricted to be of equal length.  Argument for a rectangle being a subclass of a square:  - a rectangle is a square that just has an extra attribute: the ratio between the sizes of the width and height.  Argument for neither:  - a rectangle has two attributes determining its shape and a square has just one.  If the two attributes of a Rectangle have the same value, is it equivalent to a Square object? |
| **Exercise 12.12**  a) Which of the following assignments are legal, and why or why not?   * Person p1 = new Student();   - This is legal because Student is a subclass of Person.   * Person p2 = new PhDStudent();   - This is legal because PhDStudent is a subclass of Person (because it is a subclass of Student which is a subclass of Person)   * PhDStudent phd1 = new Student();   - This is not legal, because Student is not a subclass of PhDStudent.   * Teacher t1 = new Person();   - This is not legal because Person is not a subclass of Teacher.   * Student s1 = new PhDStudent();   - This is legal, because PhDStudent is a subclass of Student.  b) Assume that the two illegal lines above are changed to:  PhDStudent phd1; Teacher t1;   * s1 = p1;   - This is not legal, because erson is not a subclass of Student. The compiler only knows the static type of p1 which is Person - it does not know the dynamic type which is Student.   * s1 = p2;   - This is not legal, because a Person is not a subclass of Student (same arguments the previous case).   * p1 = s1;   - This is legal because Student is a subclass of Person.   * t1 = s1;   - This is not legal because Student is not a subclass of Teacher.   * s1 = phd1;   - This is legal because PhDStudent is a subclass of Student.   * phd1 = s1;   - This is not legal because Student is not a subclass of PhDStudent. |
| **Exercise 12.14**  Nothing has to change in the NewsFeed class when we add a new Post subclass. This is because the NewsFeed never worries about the actual subclass, but treats all objects as Posts. |
| **Exercise 12.15**  From the JDK API documentation, the following class hierarchies can be drawn: |
| **Exercise 12.16**  See: 12-16-lab-classes |
| **Exercise 12.17**  An example of a class hierarchy of some of the components in a computer. Note that the question refers to a DVD drive rather than a CD drive, but the change is trivial.  Or maybe this: |
| **Exercise 12.18**  The legal statements tells us the following:  a) m = t This tells us that T is subclass of M  b) m = x This tells us that X is a subclass of M  c) o = t This tells us that T is a subclass of O. But T was also a subclass of M, and Java does not allow multiple inheritance. Therefore M must be a subclass of O or vice versa.  And the illegal statements gives us:  d) o = m M is **not** a subclass of O. Hence from c) we have that O is a subclass of M.  e) o = x X is **not** a subclass of O  f) x = o O is **not** a subclass of X  From this information we can be sure that the following relations exist: |
| **Exercise 12.19**  See the diagram in the solution to exercise 12.15. |