(3 points) 1) Why do we choose to implement add(Equipment e) instead of a more specific method like add(Guitar g)?

We use a single add(Equipment e) so that the inventory can handle any type of equipment without needing a separate method for each one. This makes the code more flexible and easier to maintain: we can add new equipment types in the future without writing more methods. If we wrote add(Guitar g), add(Keyboard k), etc., we'd have to add a new method every time we introduce a new instrument or furniture item. This also lets us store all equipment in a single list and treat them polymorphically.

(3 points) 2) Why do we use a String instead of an Equipment object in the HashMap?

We use the String returned by toString() to represent the type of equipment because we want to group counts by type, not by individual objects. If we used the Equipment object itself as the key, each object would be treated as a separate key, even if it's the same type. Using a String makes it simple to see that we have, for example, 3 Guitars instead of tracking every individual Guitar object.

(3 points) 3) Why do we use an Integer instead of an int in the HashMap?

HashMap in Java can only store objects as keys and values, not primitive types. int is primitive, so we use Integer, which is the object wrapper for int. This allows the count to be stored in the map and automatically handles operations like put and get. If we tried to use int, it would cause a compile-time error because HashMap cannot hold primitives directly.

(3 points) 4) Why do we need to define a custom toString() method for the Equipment types? What happens if we use the default method?

We define a custom toString() so that each subclass returns its type name, like "Guitar" or "Stool". This is important because our HashMap uses the toString() value as the key. If we used the default toString() from Object, it would return something like "Guitar@1f32e575", which is unique for each object. That would break the counting logic, since the map would treat every object as a different type, even if they are the same class.

(3 points) 5) What happens if we only define a custom toString() method for the Equipment or Instrument class (but not for any of the subclasses)?

If only the parent class defines toString(), all subclasses inherit that method. If it just returns "Equipment" or "Instrument", the HashMap would group all subclasses under the same key. For example, Guitars and Keyboards would all show up as "Instrument" with their counts combined, and Chairs and Stools would all show as "Equipment". This prevents us from distinguishing between different types of instruments or furniture, which defeats the purpose of tracking counts per type.