Robustness —

- Check for `null` parameters. An `IllegalArgumentException` is more appropriate than letting the `NullPointerException` happen when the parameters shouldn't be `null` in the first place.
- Also check that the vertex or edge being manipulated belongs to the graph in question. This requires storing some additional information to keep the check $O(1)$.
- Checking that the vertex/edge is the right type is a bit more of a judgment call — it will also be caught with a `ClassCastException` and arguably `ClassCastException` is more informative than `IllegalArgumentException` though one can also make the case that it violates encapsulation.
- Be careful that all robustness checks are within the big-Oh of the operation itself — ideally this means always $O(1)$, though a more expensive check can be OK if the operation itself is also more expensive.

Organization —

- Elements common to both adjacency list and adjacency matrix (and only those elements) should go in `AbstractGraph`, `AbstractVertex`, `AbstractEdge`. The most common problem was not pulling enough into the `Abstract*` classes — the lists of vertices and edges along with methods that only work with them should be in `AbstractGraph`, the degree and reference into the list of vertices should be in `AbstractVertex`, and the endpoints and reference into the list of edges should be in `AbstractEdge`.
- Using one method to implement another is good, but be careful not to impact the efficiency of the method. For example, `areAdjacent` should be $O(\min(\deg(u), \deg(v)))$ for adjacency list and $O(1)$ for adjacency matrix. If implemented in `AbstractGraph` using `incidentEdges`, the runtime is $O(\deg(v))$ for adjacency list and $O(|V|)$ for adjacency matrix.

Efficiency —

- `degree(`v`)` should be $O(1)$ — store the degree (and update on insert/remove edge) rather than counting the incident edges each time.
- `areAdjacent(`u`,`v`)` should be $O(\min(\deg(u), \deg(v)))$.