1 Administrivia

- No lab tomorrow
- Problem Set 3 due next Wednesday

2 Equality

There are two types of equality that we care about typically:

- **Reference equality** — are two objects the same in memory?
- **Logical equality** — do two objects represent the same value?

Reference equality is typically easy to define and use correctly, since its semantics remain the same over all types. Logical equality, however, since it can depend on the types involved, requires additional constraints in order to make sense. In order an equality implementation to be completely correct, it must:

- Define an equivalence relation (reflexive, symmetric, transitive)
- Be consistent (should not change over time)
- Imply an equal **hashCode** (if two objects are equal, they should have equal hash codes)

More specifically, the first point, defining an equivalence relation, can be broken down as:

- **Reflexive** — \( A = A \)
- **Symmetric** — \( A = B \rightarrow B = A \)
- **Transitive** — \( A = B \) and \( B = C \rightarrow A = C \)

It is hard to correctly override `equals`, especially when inheritance is involved. Some things to keep in mind:

- It is very difficult to correctly define equality when inheritance is used, consider using composition instead.
- When overriding `equals`, always override `hashCode`.
- Consider using the `@Override` annotation to make sure that you are correctly overriding `equals` and `hashCode`.
- Eclipse can automagically override `equals` and `hashCode` for you, **Source → Generate hashCode() and equals()**...

3 Hashing

Hashing is the idea of taking an arbitrary object and converting it into an integer known as a **hash code** that represents that object. This is a powerful idea for two main reasons:

- **Object identity** — If the hash codes of two objects differ, it is easy to tell that they are not the same (the converse is not necessarily true).
- **Hash tables** — We can use hashing to map objects to array indices to create a **hash table**, a very powerful and commonly used data structure.

The key intuition for hashing is that a hashing function is essentially trying to **compress** an object’s identity and uniqueness into as compact a representation as possible (an integer).