Glossary

antisymmetric

An **antisymmetric relation** is any relation such that `a.relation(b) && b.relation(a)` always implies that `a` and `b` are equal.

associative

An **associative operation** is any binary operation for which `a.operation(b).operation(c)` always equals `a.operation(b.operation(c))`.

binary operation

A binary operator (such as `+`), or a method of the form `static A operation(A first, A second)`, or a method in a class `A` of the form `A operation(A second)`.

commutative

An operation is **commutative** if `a.operation(b)` always equals `b.operation(a)`.

See symmetric for the analogous property of relations.

equal

In referring to [privitives][], `p1` and `p2` are called equal if and only if `p1 == p2`. In referring to objects, *unless specified otherwise*, `a1` and `a2` are called equal if and only if `a1.equals(a2)`. When we intend identity equality we will clearly say, "the same instance as." (Range is an example of a class that specifies a different meaning for "equal".)

equivalence relation

An **equivalence relation** is any binary relation that is reflexive, symmetric and transitive. The contract of `Object.equals` specifies that it must define an equivalence relation.

idempotent

In an **idempotent operation**, `a.operation().operation()` always equals `a.operation()`.

lazy

A **lazy view** does not query the backing object until it absolutely has to. For example, `Iterators.filter(Iterator, Predicate)` returns an `Iterator` that only advances the backing iterator when a new element is demanded.

partial ordering

A relation is said to be a **partial ordering** if it is reflexive, transitive, and antisymmetric. If it also has the property that `a.relation(b) || b.relation(a)` for all `a` and `b`, it is a total ordering.

primitive

A `boolean`, `byte`, `short`, `char`, `int`, `float`, `long` or `double`.

reflexive

In a reflexive relation, `a.relation(a)` is always `true`.

relation

A relational operator (such as `<`), or a method of the form `static boolean relation(A first, A second)`, or a method in a class `A` of the form `boolean relation(A second)`.

symmetric

In a **symmetric relation**, `a1.relation(a2)` always has the same boolean value as `a2.relation(a1)`.

total ordering

A **total ordering** is any relation that is antisymmetric, transitive, and has the property that `a.relation(b)` or `b.relation(a)` holds. (It follows that `relation` is reflexive.)

For example, when we say that a `Comparator` must define a total ordering, we mean that the relation `comparator.compare(a, b) <= 0` satisfies:

- `comparator.compare(a, b) <= 0 && comparator.compare(b, a) <= 0` does imply that `comparator.compare(a, b) == 0`, and if the ordering is *consistent with equals*, `a.equals(b)`.
- `comparator.compare(a, b) <= 0 && comparator.compare(b, c) <= 0` implies that `comparator.compare(a, c) <= 0`.
- For all `a, b`, `comparator.compare(a, b) <= 0 || comparator.compare(b, a) <= 0`.

transitive

For a **transitive relation**, `a1.relation(a2) && a2.relation(a3)` always implies that `a1.relation(a3)`.