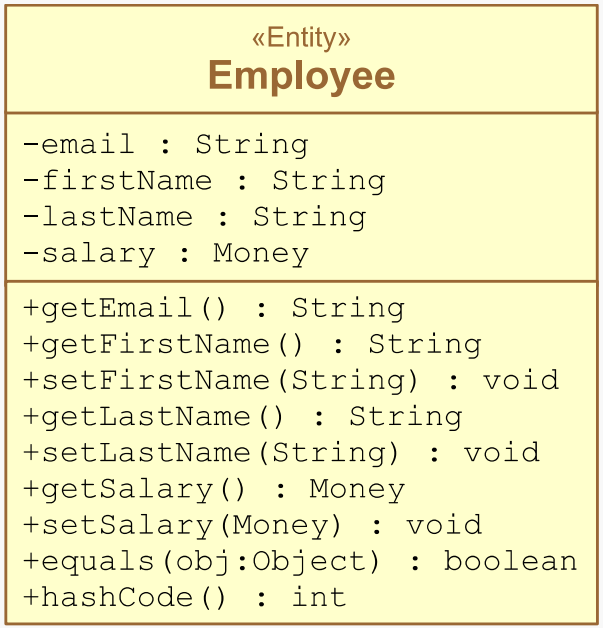
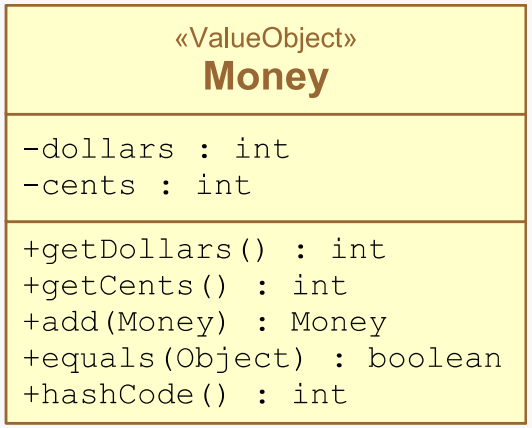
Domain-Driven Design Activity





SWEN-261
Introduction to Software Engineering

Department of Software Engineering
Rochester Institute of Technology

Software Engineering
Rochester Institute of Technology

Entities and Value Objects are special types of objects

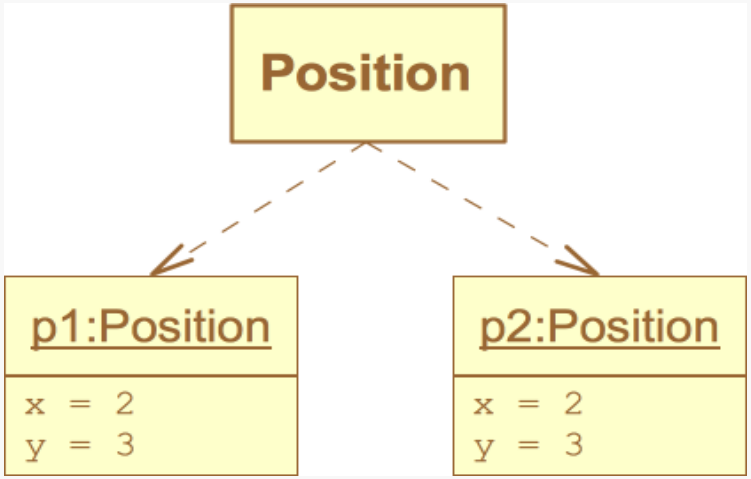
- Normal Java equality semantics are not adequate with dealing with Entities and VOs
- So, what does this mean *equality semantics*?
  - ***Good question!***
- The Java == operator only tests that the two object references are the same.
  - ***But having the "same location in the heap" is meaningless for these types of objects***
  - ***The following slides explain the equality semantics of both of these object types***
  - ***Starting with Value Objects…***

Value Objects have *value semantics*.

- Value Object components represent *values in the real world*: money, measurements, positions, and so on
- Value objects must be equal based upon the internal data of the value.
  - ***For example, a coordinate `Position` is based upon an x,y pair of integers***
- Value objects must be immutable.
  - ***Once set in a constructor no attribute may change***
  - ***No mutator methods; ie, no setters***

Value Objects are equal when their internal data are both equal

- Let's give an example:



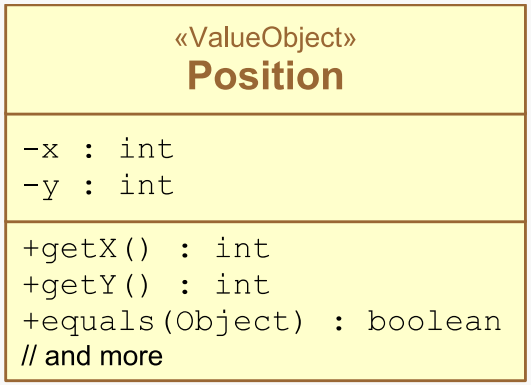

```
public void make_multiple_positions() {
  Position p1 = new Position(2, 3);
  Position p2 = new Position(2, 3);
  if (p1 != p2) {
    // The two distinct objects have different identities.
  }
 if (p1.equals(p2)) {
    // The two value objects are equal.
  }
}
```

The default behavior is the same as the == operator but that is not appropriate for value objects.

Unfortunately the default `equals` method uses reference identity. Doh!

- The solution is easy: override the `equals` method with this type's equality semantics.

```
public class Position {
  private int x;
  private int y;

  // more code here

  @Override
  public boolean equals(Object obj) {
    if (obj == this) return true;
    if (!(obj instanceof Position)) return false;
    final Position that = (Position) obj;
    return this.x == that.x && this.y == that.y;
  }
}
```



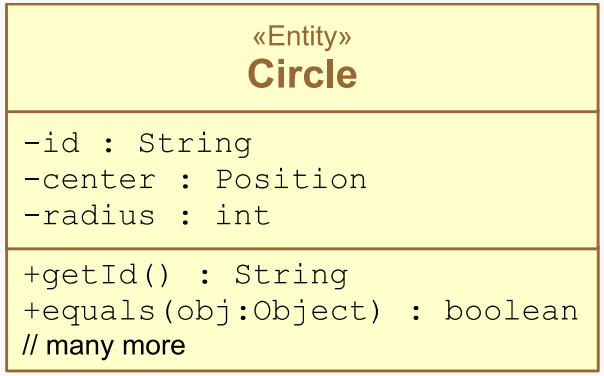
Equality is based upon all attributes.

Entities have *identity semantics*.

- Entity components represent *things in the real world*: people, orders, products, and so on
- What identifies these types of things?
- In an Enterprise application the system would store entities in a database.
  - ***The database assigns a unique ID to each entity object.***
- When you don't have a database you choose an attribute that is unique and unchanging.
  - ***This is often called a*** natural key***.***

Provide an id for an Entity class.

```
public class Circle {
  private String id;
  private Position center;
  private int radius;

  public Circle(String id) {
    this.id = id;
  }

  public String getId() {
    return id;
  }
  // more code here

  @Override
  public boolean equals(Object obj) {
    if (obj == this) return true;
    if (!(obj instanceof Circle)) return false;
    final Circle that = (Circle) obj;
    return this.id.equals(that.id);
  }
}
```



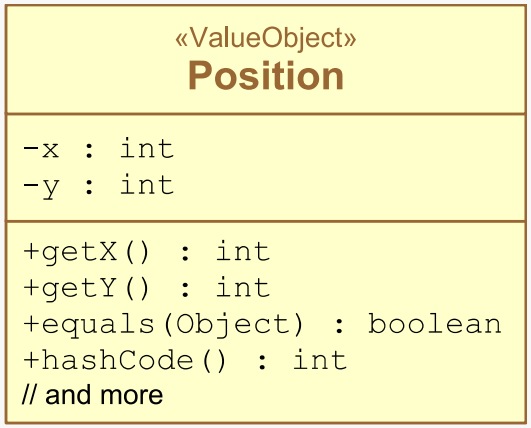
Let the client of the Circle specify a unique id.

So now that we have semantic equality, we need a semantic hash code.

- In Java there is a close relationship between the `equals` and `hashCode` methods.
  - ***If you override one you must override the other.***
  - ***Use the attributes that make up the equality check when building the hash code.***
  - ***If two objects are "equal" then they must also have the same hash code:***
    `x.equals(y)` **=>** `x.hashCode() == y.hashCode()`
- This is critical when you use objects as keys in a `HashMap` or stored in `HashSet` collections.
  - ***See Java API hashCode docs for explanation.***
  - ***See ProgramCreek blog for another explanation.***

Value Objects with primitive attributes can calculate its own hash code with simple arithmetic.

```
public class Position {
  private int x;
  private int y;

  // more code here

  @Override
  public boolean equals(Object obj) {
    if (obj == this) return true;
    if (!(obj instanceof Position)) return false;
    final Position that = (Position) obj;
    return this.x == that.x && this.y == that.y;
  }

  @Override
  public int hashCode() {
    return x * 31 + y;
  }
}
```

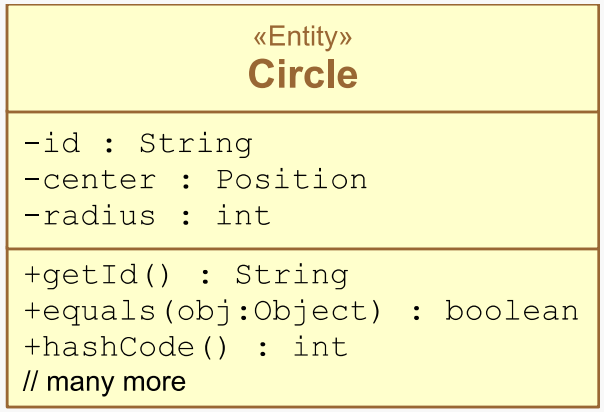


Java 8 now supplies a helper method:

```
return Objects.hash(x, y);
```

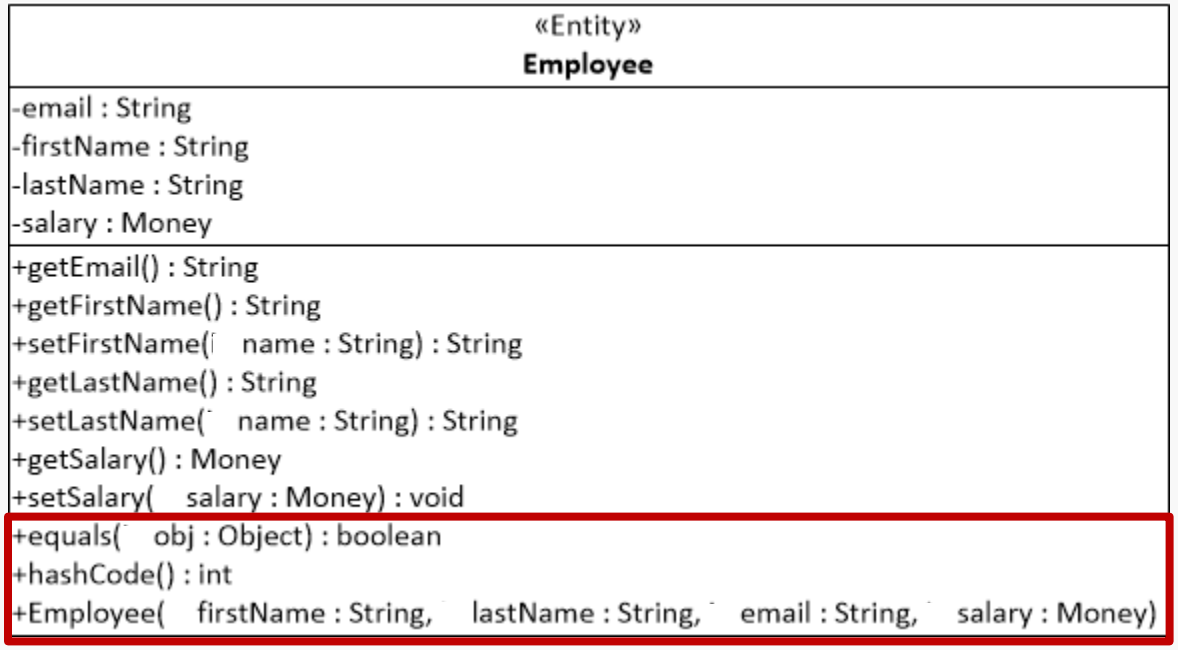
Entities should use the ID to calculate a hash code.

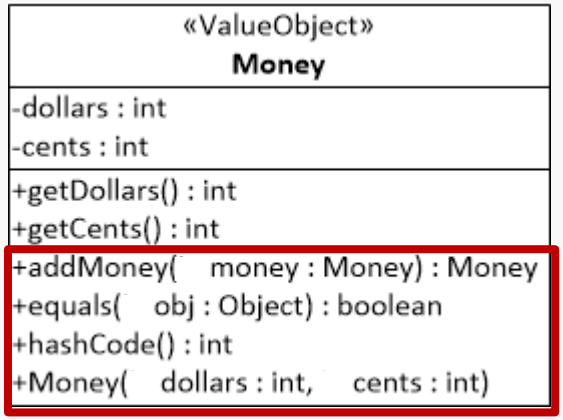
```
public class Circle {
  private String id;
  private Position center;
  private int radius;

  // more code here

  @Override
  public boolean equals(Object obj) {
    if (obj == this) return true;
    if (!(obj instanceof Circle)) return false;
    final Circle that = (Circle) obj;
    return this.id.equals(that.id);
  }

  @Override
  public int hashCode() {
    return id.hashCode();
  }
}
```



Your exercise is to build the code for this model.

- Implement the methods indicated in these two Model classes:





- The classes must have the attributes and implementations of the methods shown in red. This must compile!
- Place the two source files into a single zip file and deposit it in the *Domain-driven design - individual* dropbox.