Builders

How do we build or create an object? Usually it's a constructor. What about an alternative: use one object to create another object!

Example: `StringBuilder`

`StringBuilder` is an much more efficient way to create a complex `String`, because the `concat()` operation is inefficient it provides an `append()` that grows the string in an efficient way.

Example: `AccountBuilder`

Create a builder for `Account` object creation. First create the builder class with the fields to collect the "settings" of the builder. The constructor will set some defaults for the built object. There is usually some creativity in writing the setter names!

```
public class AccountBuilder {

    private long id;
    private String holder;
    private double balance;
    private double rate;
    private boolean charge;

    public AccountBuilder() {
        holder = "unknown";
        id = 0;
        balance = 0.0;
        rate = 0.0;
        charge = false;
    }

    public void withAccountId(long id) {
        this.id = id;
    }

    public void withOwner(String holder) {
        this.holder = holder;
    }
    public void startingBalance(double balance) {
        this.balance = balance;
    }

    public void interestRate(double rate) {
        if(rate <= 0.0 || rate > 1.0)
            throw new IllegalArgumentException();
        if(charge)
            throw new IllegalStateException("Incompatible account types.");

        this.rate = rate;
    }

    public void chargeWithdrawals() {
        charge = true;
    }

    ...
```

Then add a ”build” method, also sometimes called ”make” or ”create”:

```
    ...

    public Account create() {

        // check for incompatible settings
        if(rate > 0.0 && charge)
            throw new IllegalStateException("Incompatible account types.");

        Account account;
        if(rate > 0.0)
           account = new SavingsAccount(id, holder, balance, rate);
        else if(charge)
           account = new CheckingAccount(id, holder, balance);
        else
           account = new Account(id, holder, balance);
        return account;
    }

}
```

Here's how we can use our builder:

```
AccountBuilder builder = new AccountBuilder();
builder.withAccountId(123l);
builder.withOwner("ian");
builder.startingBalance(100.00);
builder.interestRate(0.05);
// can't have both: builder.chargeWithdrawals();
Account account = builder.create();
```

Compared to the constructor version:

```
Account account = new SavingsAccount(123l, "ian", 100.00, 0.05);
```

Benefits

Possible benefits:

1. clearer initalization than constructor: descriptive setters vs. constructor parameters.
2. less confusing that multiple constructors.
3. simplify (or make more efficient) the initialization of a complex object. (Ex: `StringBuilder`).
4. can use the builder to make instances of the sub-classes of a super-class.

Builder Setters

Traditional setters are like this:

```
class Point {

    private int x;

    public void setX(int x) {
        this.x = x;
    }

}
```

whereas a builder setter is like this:

```
class Point {

    private int x;

    public Point setX(int x) {
        this.x = x;
        return this;
    }

}
```

Why? To chain together setters!

For example here’s how we can change the previous example to use builder setters:

```
AccountBuilder builder = new AccountBuilder();
builder.withAccountId(123l);
builder.withOwner("ian");
builder.startingBalance(100.00);
builder.interestRate(0.05);
Account account = builder.create();
```

Since each setter call returns `builder` we can chain them together:

```
AccountBuilder builder = new AccountBuilder();
builder.withAccountId(123l).withOwner("ian").startingBalance(100.00).interestRate(0.05);
Account account = builder.create();
```

And since `new AccountBuilder()` is the same as `builder` we can include it in the chain:

```
AccountBuilder builder = new AccountBuilder().withAccountId(123l).withOwner("ian").startin
Account account = builder.create();
```

Since Java does not consider whitespace part of the syntax we can make this more appealing with some newlines:

```
AccountBuilder builder = new AccountBuilder()
    .withAccountId(123l)
    .withOwner("ian")
    .startingBalance(100.00)
    .interestRate(0.05);
Account account = builder.create();
```

Finally, notice the `builder` variable can be removed since it's only created for the call to `create()`. Now we get this very standard builder statement:

```
Account account = new AccountBuilder()
    .withAccountId(123l)
    .withOwner("ian")
    .startingBalance(100.00)
    .interestRate(0.05)
    .create();
```

Exercise: create a builder for survey questions.

Using the survey exercise, complete the builder `QuestionBuilder` by implementing the `build()` method.

Videos:

- `https://youtu.be/oT2TucqsZPU`