Back to BankAccount:

Another important method: toString().

//returns string of info about this object
public String toString()
{
    return this.acctNum + "" + this.acctName + "" +
            this.balance;
}

Back to Driver:

System.out.println(acct1);

(instead of: printIn(acct1.toString()).

Output: "12345" + Donald + "" - 50

Sometimes may want different versions of the same method
ex: withdraw() that always withdraws $20.

method overloading - using the same name for multiple
methods, (How will the compiler know which
method to call?) parameter lists must be
different
BankAccount:

```java
public class BankAccount {

    private final double AMOUNT = 20;
    // value can't be changed

    public void withdraw() {
        this.balance -= AMOUNT;
    }

    AMOUNT = 30; // won't compile!
}
```

This was simplest approach, now more sophisticated.

Objects provide abstraction - we know they do something
but we don't know how (implementation details).

ex. Scanner: nextInt() String: charAt()

Only the object's class deals with implementation
details.

Encapsulation - an object should be modifiable only by
the object's methods.
Driver should not have direct access to object's data.

To enforce, use visibility modifiers.

Visibility modifiers - enforce the accessibility of an object's data fields/methods.

- public - accessible by any class within same project (by default).
- private - only within object's class (another one we won't use): protected

To encapsulate a class:

- make all data fields private
- How to access from outside class?
- create public "setter" + "getter" methods for each data field
BankAccount now:

private String acctNum;

//setter for acctNum
public void setNum(String num) {
  this.acctNum = num;
}

//getter for acctNum
public String getNum() {
  return this.acctNum;
}

Driver now:

acct1.acctNum = "12345" => acct1.setNum("12345")

println(acct1.getNum());

Why encapsulate?

1. Prevent need to modify Driver
   Suppose later decide acctNum should be int
   with encapsulation, just need to change class, not driver

2. Enforce validity of data fields.
   Suppose acctNum should be 9-digit
   Can't enforce with acct1.acctNum =
   Can " " acct1.setAcctNum( )