Recap: Data types in Java:
- primitive data: int, double, char, boolean
- built-in objects: String, Scanner, arrays, Random
- programmer-defined objects

Methods/Operations
- primitive: +, -, =, <=, <
- built-in: equals(), nextInt(), [ ]
- programmer-defined: replace And Count()

Object - entity with attributes (data fields) and operations (methods)
Class - defines data fields + methods of an object

Ex. String str = "hello";
        ↑   ↑   ↑
        class    object    value
                    (data field)

Another data field? length
Methods: replace(), toUpperCase(), equals()

Classic example: programmer-defined object: Bank Account

First step: Think about which attributes + operations would be useful.
attributes/data fields operations/methods (return type? parameters
balance (double) withdraw()
account number (String) deposit()
name (String) checkBalance()"
public class BankDriver

    public static main()

    BankAccount acct1 = new BankAccount("12345");

        acctNum = "12345"
        name = "Drumpy"
        balance = 100

    acct1, name = "Drumpy",
    acct1, balance = 100;

Back to BankAccount class.

Write the methods:

    method header now:
    public static return type name (parameter-list)

    public double getBalance()
        return this.balance;
    
    public void withdraw (double amt)
        this.balance -= amt;
public void deposit(double amt) {
    this.balance += amt;
}

Back to BankDriver:

acct1.deposit(50);
acct1.withdraw(200);

balance = 100 150 - 50

acct1

Back to BankAccount:

"Can also create an empty constructor:

public BankAccount()

"?

?

Back in Driver:

BankAccount acct2 = new BankAccount();
acct2.acctNum = "56789";
acct2.name = "";
acct2.balance = 0.0
Back to BankAccount:

Another important method: toString()

// returns String of info about this object
public String toString()
{
    return acctNum + " " + name + " " + 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 $120.

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 {
    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.