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(), [], nextInt()
- Programmer-defined: replace, And, Count()

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

Terms also apply to built-in objects.

```
Ex: String str = "hello";
   ↑ ↑ ↑
   class object/ value
       variable
```

Attribute: length
Method: replace(), toUpperCase(), equals()...

Classic example: programmer-defined object: Bank Account

First step: Think about which attributes + operations would be useful.
attributes

- balance (double)
- account number (String)
- name (String)

operations/methods (return type? parameters)

- withdraw()
- deposit()
- checkBalance()

Simplified approach first:

```java
public class BankAccount { save as BankAccount.java

    String acctNum, name;  // global scope
    double balance

    // constructor - for creating a BankAccount object
    public BankAccount(String num )
    {
        acctNum = num;  // typically an attribute
                        // that is unique to
class Driver

    this acctNum = num;  // each object

    ```
public class BankDriver

public static main()

BankAccount acct1 = new BankAccount("12345");

acct1.name = "Drumpf"
acct1.balance = 100

acct1.name = "Drumpf"
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.withdraw(40);
    acct1.deposit(200);
    balance = 100, 60, 260
    acct1

Back to BankAccount:

    Can also create an empty constructor:

    Public BankAccount() {

Back in Driver:

    BankAccount acct2 = new BankAccount();
    acct2.acctNum = "56789"
    acct2
    acct2
    acct2
Back to `BankAccount`:

Another important method: `toString()`

```java
public String toString()
{
    return acctNum + " " + name + " " + balance;
}
```

Back to `Driver`:

```java
System.out.println(acct1);
```

(instead of `printIn(acct1.toString());`)

Output: "12345" + " Drumpf" + " 260"

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:

public class BankAccount {

    final double AMOUNT = 20;

    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.