What You Know So Far

- Each object has its own copy of methods and fields:

```java
class BankAccount {
    private String name;
    private double balance;
    public void withdraw(double amount) {

    }
}

BankAccount mikeAcc = new BankAccount("Mike", 100);
BankAccount zachAcc = new BankAccount("Zach", 20);
```
Instance Fields and Methods

• Each object has its own copy of methods and fields:

```
MikeAcc
- String name
- double balance
- void setName(String)

ZachAcc
- String name
- double balance
- void setName(String)
```
Instance Fields and Methods

BankAccount mikeAcc = new BankAccount("Mike", 100);
BankAccount zachAcc = new BankAccount("Zach", 20);

System.out.println(mikeAcc.getBalance()); //100
System.out.println(zachAcc.getBalance()); //20

zachAcc.withdraw(19);

System.out.println(mikeAcc.getBalance()); //100
System.out.println(zachAcc.getBalance()); //1
Shared Fields

- What if we wanted to make a field shared among all objects of a class?

BankAccount Class
- double interestRate

mikeAcc
- String name
- double balance
- void setName(String)

zachAcc
- String name
- double balance
- void setName(String)
Static Fields

- A given class will only have one copy of each of its static fields
  - This will be shared among all the objects.

- Each static field exists even if no objects of the class have been created.

- Use the word `static` to declare a static field.
Static Fields

- Only one instance of a static field data for the entire class, not one per instance.

- "static" is a historic keyword from C/C++
Static Fields Example

class BankAccount {
    public static double interestRate = 0.02;
}

BankAccount mikeAcc = new BankAccount("Mike", 100);
BankAccount zachAcc = new BankAccount("Zach", 20);

System.out.println(mikeAcc.interestRate); // 0.02
System.out.println(BankAccount.interestRate); // 0.02

mikeAcc.interestRate = 0.05;
System.out.println(zachAcc.interestRate); // 0.05
public class BankAccount {

private static int numAccounts = 0;

public BankAccount(String name, double balance) {
    numAccounts++;
}
}
Unique ID for Objects

```java
public class BankAccount {
    private static int nextAccountNum = 0;
    private int accountNum;

    public BankAccount(String name, double balance) {
        accountNum = nextAccountNum++;
    }
}
```
public class BankAccount {
    private static BankAccount[] accounts =
        new BankAccount[100];
    private static int nextAccountNum = 0;

    public BankAccount(String name, double balance) {
        accounts[nextAccountNum++] = this;
    }
}

What would happen if we deleted this static modifier?
public class BankAccount {
    private BankAccount[] accounts =
        new BankAccount[100];
    private static int nextAccountNum = 0;

    public BankAccount(String name,
                        double balance) {
        accounts[nextAccountNum++] = this;
    }
}
More Static Field Examples

Constants used by a class:

- Usually used with `final` keyword
- Only need to have one per class; don’t need one in each object:
  ```java
  public static final double TEMP_CONVERT = 1.8;
  ```
- If variable `TEMP_CONVERT` is in class `Temperature`, it is invoked by:
  ```java
  double t = Temperature.TEMP_CONVERT * temp;
  ```
Instance Methods

- These are what you know so far...

- These define the operations you can perform on objects of a class.

- Methods typically operate on the instance (non-static) fields of the class.
  - Each object has a “copy” of the method just as it has copies of the fields.
Static / Class Methods

- Static methods are shared by all objects of the class
- One copy for all objects

BankAccount **Class**

- `int numAccounts`
- `int getNumAccounts()`

mikeAcc

- `String name`
- `double balance`
- `void setName(String)`

zachAcc

- `String name`
- `double balance`
- `void setName(String)`
Static Methods

To define a class method, add the keyword `static` to its definition.

```java
public class BankAccount {
    private static int numAccounts = 0;

    public static int getNumAccounts() {
        return numAccounts;
    }
}
```
public class BankAccount {
    private static int numAccounts = 0;

    public static int getNumAccounts() {
        return numAccounts;
    }
}

BankAccount mikeAcc = new BankAccount("Mike", 100);
System.out.println(mikeAcc.getNumAccounts()); //1

BankAccount zachAcc = new BankAccount("Zach", 20);
System.out.println(mikeAcc.getNumAccounts()); //2
System.out.println(BankAccount.getNumAccounts()); //2
Static Methods

• Static methods do not operate on a specific instance of their class

• Have access only to static fields and methods of the class
  – Cannot access non-static ones
Static Methods Limitations

```java
public class BankAccount {
    private static int nextAccountNum = 0;
    private int accountNum;

    public static int getAccountNum() {
        return accountNum;
    }
}
```

Illegal, cannot access non-static field from static method
More Static Methods

- Static methods are also used when you need to define a method on 2 objects.

```java
public static BankAccount greaterBalance
    (BankAccount bal1, BankAccount ba2)
{
    if (bal1.balance() >= ba2.balance())
        return bal1;
    else
        return ba2;
}
```
Static Method Examples

- For methods that use only the arguments and therefore do not operate on an object
  
  ```java
  public static double pow(double b, double p)
  // Math class, takes b to the p power
  ```

- For methods that only need static data fields

- We **HAVE TO** use the static key word on the `main` method in the class that starts the program
  - No objects exist yet for the main method to operate on!
The `final` keyword

- Sometimes you will declare and initialize a variable with a value that will never change.

- To prevent any accidental changes, Java provides you with a way to fix the value of any variable by using the `final` keyword when you declare it.
The `final` keyword

- We declared `PI` as
  ```java
  public static double PI = 3.14159;
  ```
  but this does not prevent changing its value:
  ```java
  MyMath.PI = 999999999;
  ```

- We use keyword `final` to denote a constant:
  ```java
  public static final double PI = 3.14159;
  ```

- Once we declare a variable to be `final`, it's value can no longer be changed!
Final References

• Consider this final reference to a `Point`:
  ```java
  public static final Point ORIGIN =
      new Point(0, 0);
  ```

• This prevents changing the reference `ORIGIN`:
  ```java
  MyMath.ORIGIN = new Point(3, 4);
  ```

• **BUT!** You can still call methods on `ORIGIN` that change the state of `ORIGIN`.
  ```java
  MyMath.ORIGIN.setX(4);
  ```