Static Methods

A static method belongs to a class.

It isn't part of any object.

You invoke a static method using the class name.

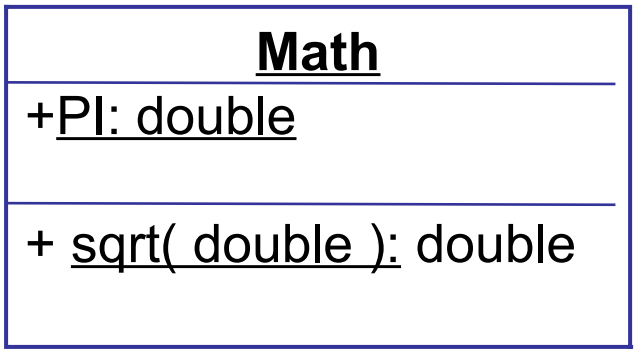
```
double root = Math.sqrt( 5.0 );
// a static method of Math class
char c = 'a';
if ( Character.isLetter( c ) )
  // static method in Character class
   System.out.println(c + " is a letter");
```

Meaning of static Methods

Most static methods can be thought of as ***services*** provided by the class.

- `Math.sqrt( )` is a service of the Math class.
- `Character.toUpperCase( c )` is a service of the Character class.

Static Methods: UML class diagram



Writing a static method

Just put the word "static" in the header.

```
class BankAccount {

   private static double getInterestRate( ) {

   ... }


   public static void main( String [] args ) {

   ... }
```

static method cannot access instance attributes

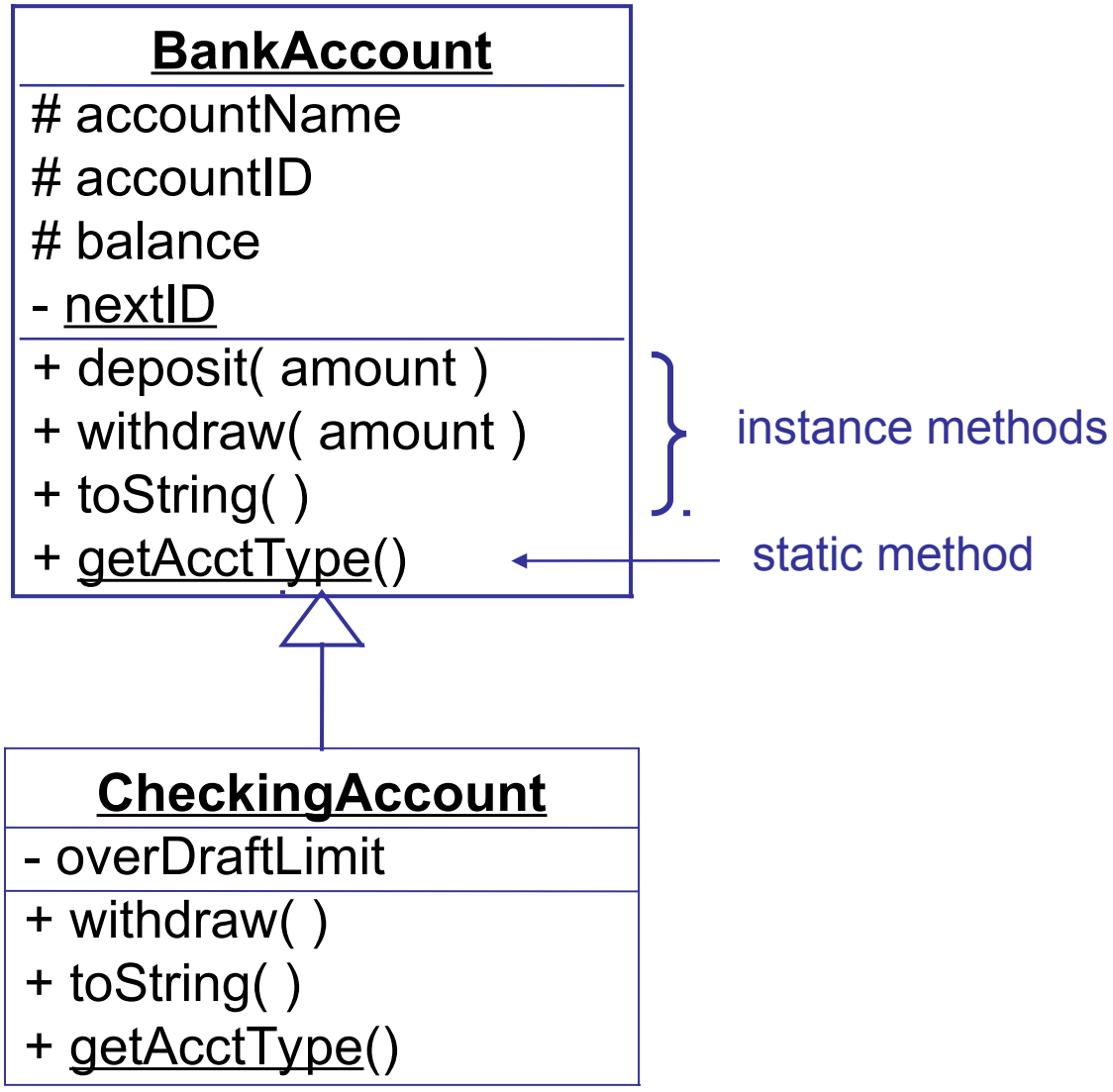
No object == no attributes.

```
class BankAccount {
    private double balance;
    public static void main(String [] args) {
        // ERROR - main() is static but balance
        // is an instance attribute
        int sum = balance; // ERROR
        BankAccount acct = new BankAccount( );
        // OK - access attribute through object
        int sum = acct.balance; // OK
    }
```

Binding of Static Methods

- Static methods are *statically bound*. That is, the compiler decides which method implementation to use.
- An object of a subclass can exhibit static behavior from a parent. But it can be tricky...
  - For ***instance methods***, the method called depends on the type of the ***object*** *(determined at run-time).*
  - For ***static methods***, the method called depends on the type of the ***object reference*** *(determined at compile-time).*

Static Methods: UML class diagram



Instance Methods are Polymorphic

Consider the instance method **`toString()`**.

Which toString() is called in each case?

```
BankAccount acct = new BankAccount("Plain");
BankAccount chck = new CheckingAccount("Checking");
System.out.println( "acct=" + acct.toString() );
System.out.println( "chck=" + chck.toString() );
```

```
acct=[BankAccount] Plain 00000001

chck=[CheckingAccount] Checking 11000002
```

Instance Methods

The toString() method that is invoked depends on the actual type of the object reference (determined at run-time).

Static Methods

Test behavior using the static **`getAccountType`**().

```
public class BankAccount {

  public static String getAcctType() {
    return "Bank Account";
  }
```

```
public class CheckingAccount
                    extends BankAccount {

  public static String getAcctType() {
    return "Checking Account";
  }
```

Static Methods are Not Polymorphic

You *shouldn't* invoke static behavior using object references, but sometimes you'll see this:

```
BankAccount acct = new BankAccount("Plain");
BankAccount chck = new CheckingAccount("Checking");
System.out.println( "acct=" + acct.getAcctType() );
System.out.println( "chck=" + chck.getAcctType() );
```

```
acct=Bank Account

chck=Bank Account
```

Accessibility

Static context (inside a **static method**)	**Instance** context (inside an **instance method**)
can only access static elements, unless you use a reference to an object, .e.g. Student s = new Student(...); s.name; // OK	can access both instance and static elements. "elements" means attributes and methods.