Use inheritance to factor out common functionality

<table>
<thead>
<tr>
<th>Undergraduate</th>
<th>Graduate</th>
</tr>
</thead>
<tbody>
<tr>
<td>first: String</td>
<td>first: String</td>
</tr>
<tr>
<td>last: String</td>
<td>last: String</td>
</tr>
<tr>
<td>academicClass: String</td>
<td>academicYear: int</td>
</tr>
<tr>
<td>+displayTranscripts(): void</td>
<td>+displayTranscripts(): void</td>
</tr>
<tr>
<td>+toString(): String</td>
<td>+toString(): String</td>
</tr>
</tbody>
</table>
Factor common features into Student

Abstract class: Cannot instantiate

output student's name

override to add subclass's added field to the output

```java
public abstract class Student {
    private String first;
    private String last;

    public abstract String displayTranscripts();
    public abstract String toString();
}
```

```java
public class Undergraduate extends Student {
    private String academicClass;

    public String toString() {
        return super.toString() + academicClass;
    }
}
```

```java
public class Graduate extends Student {
    private int academicYear;

    public String toString() {
        return super.toString() + academicYear;
    }
}
```
super gives you access to the superclass's members

```
public class Student {...}
```

```
public class Undergraduate extends Student {...}
```

```
public class Graduate extends Student {...}
```
A subclass constructor calls a superclass constructor

// In subclass Undergraduate
public Undergraduate() {
    this("unsigned", "unsigned", "unsigned");
}

// In subclass Undergraduate
public Undergraduate(String first, String last, String academicClass) {
    // Calls super() implicitly if no super call made
    super(first, last);
    this.academicClass = academicClass;
}
Override a method when you want to do your own thing

// In subclass Undergraduate

public String toString() {
    return super.toString() +
    "\nAcademic class: " +
    academicClass;
}
Overload a method when you want to add flexibility

```
Time
- hr: int
- min: int
+ toString(): String
```

```
Time2
Type to enter text
+ toString(format24: boolean): String
```
Polymorphism is built into the Java psychic

...  
processStudent(new Undergraduate("Allen", "Apple", "Freshman"));
processStudent(new Graduate("Bill", "Broccoli", 1));

public void processStudent(Student student) {
    System.out.println(student.toString());
}

Q? Which toString() method gets executed? In the superclass or subclass?

A: Through polymorphism the subclass method gets executed. Here the object's type (Undergraduate or graduate) dictates, rather than the object's reference type (Student)
Dynamic binding means a called method is chosen at run time
instanceof allows you to find out the object's type at run time

```java
public void processStudent(Student student) {
    if (student instanceof Undergraduate) {
        System.out.print("Processing an undergraduate: ");
        System.out.println(((Undergraduate) student).getAcademicClass());
    } else if (student instanceof Graduate) {
        System.out.print("Processing a graduate: ");
        System.out.println(((Graduate) student).getAcademicYear());
    }
}
```
There are four accessor modes in Java:

1. Private
2. Default
3. Protected
4. Public
Private member in A accessible only in defining class

public class A {}

public class B {}

public class C {}

public class D {}

public class E {}

package a

package b
Default member in A accessible in same package

public class A {}

public class B {}

public class C {}

public class D {}

public class E {}

package a

package b
Protected member in A accessible in same package and subclasses
Public member in A accessible by all

```
public class A {}
public class B {}
public class C {}
public class D {}
public class E {}
```
To prevent extending a class use `final`

```java
public final class classThatCantBeExtended {
}
```
To prevent overriding a method use `final`

```java
public final void methodThatCantBeOverridden() {
}
```