Equals and abstract
Ch 5.3 & 5.5
Announcements

Vikas Kumar's office hour change
It was:
   Tuesday 11am-noon, 2-209

Starting next week:
   Monday 10:15 - 11:15, Keller 5-208
Highlights

- Built in functions

```java
o.toString();
o.equals(o);
```

- Abstract

```java
abstract public double distance();
```
Basic point class

Suppose we wanted to make a simple class to represent an (x,y) coordinate point and a 3D point (x,y,z)

How would you make these?

(See: Point.java and Point3D.java)
equals

Remember you should not use "==" when comparing equality between classes.

This will simply compare if the references are the same, not the actual data.

To compare equality .equals() should be used.

You need to write this for your objects (See: BadEquals.java)
Let us go back to our Point.java example:

Question: How do we determine if two points are the same or not?

Answer: If they have the same x and y points

Let's try and write and equals for two points: (See: PointEquals.java)
equals

But you cannot override the equals() as it has an input of type Object and not Point...

If you use the proper argument (Object type), then you will need to typecast (downcast)

(See: PointEqualsV2.java)

However, this will cause the program to crash if you give it weird types...
You could try to use the keyword: `instanceof` if (object `instanceof` Class) is true, then you can do the following typecast (no error): `(Class)object` (See: PointEqualsV3.java)

Unfortunately, we can typecast children into their parent's class types and can cause equals to be asymmetric
equals final

To ensure these both return the same result:

```java
a.equals(b);
b.equals(a);
```

... we can use `getClass()` instead of `instanceof`

class() will return the type of that object

(See: PointEqualsV4.java)
Remember: **super** allows you to use your parent's version of a method/object.

You cannot access the parent's parent (**super.super** is illegal)

Often useful if you want to expand an existing method to do a little more

*(See: Point.java and Point3D.java)*
Abstract classes

Remember classes are templates that we use to create objects

We can also use classes as templates for other classes (this is inheritance)

Sometimes when making these templates, we don't know what to fill in, but we want every class to have that method...
(See: BadCards, BadBirthday, BadHoliday.java)
Abstract classes

It is possible to create templates even when we don't know how some methods are going to work yet.

To do this we use an abstract class (shown below):

```java
public abstract class GoodCard {
```

To create the abstract class, you simply put the word abstract in your class declaration.
Abstract methods

Inside abstract classes, you can make abstract methods (an example shown below):

```
public abstract void insideMessage();
```

You do not define abstract methods, instead you simply put a semicolon.

Children of this class must override these methods (or be an abstract class itself).

(See: GoodCards, GoodBirthday, GoodHoliday.java)
Non-private abstract methods

Abstract methods must be inside an abstract class

Since children have to override these methods, they cannot be private (non-inherited)

```
private abstract void errorMethod();
```

Error!
Using abstract classes

It is illegal to try and create an instance of an abstract class, since it is not fully defined.

You can however, use it as a type to store its children.

In fact, abstract classes are only really useful as a parent to other classes.
A larger example

A good example of where abstract classes make sense is biological classification.

All animals are broken down into: Kingdoms, Phylum, Class, Order, Family, Genus and Species.

Abstract classes make sense, because no real animal exists until the “Species” point (until then it is categorization).
A larger example

Let's look at birds!

Class: Aves

Family: Spheniscidae

Species: Fairy Penguin, Loon