AP Computer Science A
Unit 7: Remaining Topics

Day 5:
Polymorphism/Typecasting

Apr 19-1:15 PM

OVERRIDING METHODS

As we talked briefly about last lesson, children can access methods in their parent class - in fact, if we create a dog object:

```
Dog d1 = new Dog("Fido", "Mutt", 5, 12.5);
```

We can do the following

```
d1.getAge();
```

even if getAge is coded in the Animal method. Java just keeps looking up the through the parent classes until it finds the method.

Data Encapsulation

Children do NOT have direct access to their parent's PRIVATE instance variables.

As you saw last class, we had to call the parent's constructor using super(...); to set the parent's instance variables.

Also subclass can only access or modify the **private** instance variables of its parent class by using the **accessor** and **modifier** methods of the parent class.

Mar 2-3:43 PM

Overriding Method: Child class re-defines a method from the parent class

Overriding is unlike Overloading because both methods *must have the exact same parameter list.*

If an overriden method exists in a child, objects created of child type will automatically ONLY do that one (unless there is a CALL to the super's INSIDE the child method). The correct method is chosen because, in Java, calls are determined by the type of the *actual object*, not the type of *object reference*.

This is an example of **Polymorphism**

Polymorphism is when more that one child extends the same parent and each child object is allowed to act in different way than its parent and siblings

Aug 31-11:14 AM

say we added the following method to the pet class

```
public class Pet extends Animal{
     public void MakeNoise ( ) {
        System.out.println("GRRRR");
     }
```

What would output given the following code?

```
Dog d1 = new Dog("Rover", "Mutt", 3, 27);
d1.MakeNoise();
```

If we changed the dog class as follows:

```
public class Dog extends Pet{
     public void MakeNoise ( ) {
        System.out.println("Bark! Bark!");
     }
```

Now what would output?

```
d1.MakeNoise();
Pet p1 = new Pet("Pinky", 5, 11);
p1.MakeNoise();
Animal a1 = new Animal(2,26);
a1.MakeNoise();
```

Jan 30-9:02 AM

Remember that child types can be stored in parent variables

```
Pet p1 = new Dog("Ziggy", "Pug", 8, 17);
//what do you think would print out here?
p1.MakeNoise();
```

```
Animal a1 = new Dog("Rascal", "Greyhound", 8, 17);
//what do you think would print out here?
a1.MakeNoise();
```

Jan 30-9:02 AM

Overriding Methods from the Object class

In Java, all classes are part of an immense hierarchy, with the **Object** class at the root.

As we've already seen, if we don't code the toString() method in any object we create, it prints the memory address of any object we create of that type. That is what the **Object** class tells it to do.

Last unit you learned that if you include the method:

public String toString()

Java would automatically print the returned String inside any System.out.println(...) of the object.

Aug 31-11:14 AM

11.7 The equals method

We have seen two ways to check whether values are equal: the == operator and the `equals` method. With objects you can use either one, but they are not the same.

- The == operator checks whether objects are **identical**; that is, whether they are the same object.
- The `equals` method checks whether they are **equivalent**; that is, whether they have the same value.

The definition of identity is always the same, so the == operator always does the same thing. But the definition of equivalence is different for different objects, so objects can define their own `equals` methods.

Feb 2-12:14 PM

time.java

```
public class Time {
    private int min;
    private int hour;
    public Time(int h, int m) {
        min = m;
        hour = h; }
    public int getMin () {
        return min; }
    public int getHour () {
        return hour;}
    public void setHour (int h) {
        hour=h;}
```

Feb 3-10:02 AM

runner.java

```
Time reservation1 = new Time (7, 30);
Time reservation2 = new Time (7, 30);
Time reservation3 = reservation1;
if (reservation1 == reservation2)
    System.out.print("equal");
if (reservation1 == reservation3)
    System.out.print("equal");
reservation3.setHour(5);
if (reservation1.equals(reservation2))
    System.out.print("equal");
if (reservation1.equals(reservation3))
    System.out.print("equal");
```

Feb 3-10:02 AM

instanceof and Typecasting

Why would we put child classes into parent variable types?

1. To hold one arraylist of parent type

```
ArrayList <Animal> kennel = new ArrayList <Animal> ();
kennel.add(new Dog());
kennel.add(new Cat());
kennel.add(new Bird());
kennel.add(new Dog());
```

we can loop through list and call any method from parent (or overriden from parent)

```
for (Animal a: kennel)
    System.out.println(a.getAge());
```

Feb 25-9:01 AM

If we were calling a method that only SOME types of children have, we would need to check for that type of child and then typecast appropriately:

java has an instanceof operator that lets us check

```
for (int i = 0; i < kennel.size(); i++) {
    if (kennel.get(i)  instanceof  Dog)
        ((Dog) kennel.get(i)).bark();
}
```

Feb 23-3:26 PM

2. to have generic method that can take ANY member of that "family" of classes

```
/* constructor and other methods not shown...*/
    public class Musicalinstrument{
        public void play() { }
    }
    public class drum extends Musicalinstrument {
        public void play( )  {
            System.out.println("bam, bam, bam";    }
    }
    public class trumpet extends Musicalinstrument {
        public void play( )  {
            System.out.println("toot, toot, toot";  }
    }
```

Aug 31-11:14 AM

```
public void startTheMusic(Musicalinstrument x){
        x.play();
}
in main:
    trumpet t1 = new trumpet();
    startTheMusic(t1);
    drum d1 = new drum();
    startTheMusic(d1);
```

Mar 8-12:44 PM

Mar 10-6:55 PM