## CS/ENGRD 2110
## SPRING 2019

### Lecture 6: Consequence of type, casting; function equals

#### Reminder:
- A1 due tonight

### Today’s topics
- Casting, object-casting rule
- Compile-time reference rule
- Quick look at arrays
- Implementing equals, method getClass
- Review on your own if you need to: while and for loop

### Classes we work with today

<table>
<thead>
<tr>
<th>Animal</th>
<th>subclases Cat and Dog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put components common to animals in Animal</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Animal</th>
<th>Cat</th>
<th>Dog</th>
</tr>
</thead>
<tbody>
<tr>
<td>toString()</td>
<td>purrs()</td>
<td></td>
</tr>
<tr>
<td>age</td>
<td>isOlder(Animal)</td>
<td></td>
</tr>
</tbody>
</table>

#### Casting objects

- You know about casts like:
  - `(int) (5.0 / 7.5)`
  - `(double) 6`

- You can also use casts with class types:
  - `Animal pet1 = new Cat(5);` // cast implicit
  - `Cat pet2 = (Cat) pet1;`

- A class cast doesn’t change the object. It just changes the perspective: how it is viewed!
Explicit casts: unary prefix operators

Object-casting rule: At runtime, an object can be cast to the name of any partition that occurs within it—and to nothing else. a0 can be cast to Object, Animal, Cat. An attempt to cast it to anything else causes a ClassCastException.

- a0 can be cast to Object, Animal, Cat.
- (Cat) c
- (Object) c
- (Cat) (Animal) (Cat) (Object) c

The object does not change. The perception of it changes.

Implicit upward cast

```
public class Animal {
    private int age;
    public boolean isOlder(Animal h) { return age > h.age; }
}
```

```
public class Cat {
    public String toString() { return "blinders"; }
    public void purrs() {
    }
}
```

```
public class Dog {
    public String toString() { return "blinders"; }
    public void purrs() {
    }
}
```

```
Animal pet1 = new Animal(5);
Dog pet2 = new Dog(6);
if (pet2.isOlder(pet1)) {...}
```

```
Cat c = new Cat(5);
```

Compile-time reference rule (v1)

From a variable of type C, can reference only methods/fields that are available in class C.

```
Animal pet1 = new Animal(5);
int m = pet1.purrs();
```

Illegal. The compiler will give you an error.

Checking the legality of pet1.purrs();:
Since pet1 is an Animal, purrs is legal only if it is declared in Animal or one of its superclasses.

From an Animal variable, can use only methods available in class Animal.

Quiz: Which references are legal?

- A. h.toString() OK—it’s in class Object partition
- B. h.isOlder(...) OK—it’s in Animal partition
- C. h.purrs() ILLEGAL—not in Animal partition or Object partition

Arrays
Array elements may be subclass objects

Animal[] v;
// declaration of v
v = new Animal[3];  // initialization of v
v[0] = new Cat(5);  // initialization of 1st elem
v[2] = new Dog(6);  // initialization of 2nd elem

The type of v is Animal[]
The type of each v[k] is Animal

Sometimes use horizontal picture of an array:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>2</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

Animals objects

Compile-time reference rule (CTRR), applied

Animal[] v;
// declaration of v
v = new Animal[3];  // initialization of v
Cat pet1 = new Cat(5);  // initialization of pet1
v[0] = pet1;  // initialization of 1st elem
int m = v[0].purrs();  // is this allowed?

Not allowed!
Type of v[e] is Animal.
CTRR: May reference only methods available in Animal.
purrs is not declared in Animal or one of its superclasses.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>1</td>
<td>null</td>
<td>null</td>
</tr>
<tr>
<td>2</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>

Bottom-up / Overriding rule
says function toString in Cat
partition

Equals

How Object defines equals(o)

```java
public boolean equals(Object o) {
    return this == o;
}
```

How to use equals() method
Defining equality for your own class

- **Specification:** Object.equals has a specification you must obey: reflexive, symmetric, transitive

  - Reflexive: `x.equals(x)`
  - Symmetric: `x.equals(y) iff y.equals(x)`
  - Transitive: if `x.equals(y)` and `y.equals(z)` then `x.equals(z)`

(Provided x and y are not null)

equals should say that x and y are equal iff they are indistinguishable

Function getClass and static field class

Instance method getClass() returns the class of the lowest partition in the object

```java
h.getClass() == Cat.class
h.getClass() != Animal.class
h.getClass() != Object.class
```

Equals in Animal

```java
public class Animal {
    private int age;
    /** return true iff this and obj are of the same class and their age fields have same values */
    public boolean equals(Object obj) {
        if (obj == null || getClass() != obj.getClass()) return false;
        Animal an = (Animal) obj;
        return age == an.age;
    }
}
```

Almost every method equals that you write will have these three pieces

Equals in Animal

```java
public class Animal {
    /* return true iff this and obj are of the same class and their age fields have same values */
    public boolean equals(Object obj) {
        if (obj == null || getClass() != obj.getClass()) return false;
        Cat cob = (Cat) obj;
        return purr.equals(cob.purr);
    }
}
```

Object.equals

```java
public class Point {
    public int x;
    public int y;
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }
}
```

DEMO

Almost every method equals that you write will have these three pieces
public class Point {
    /** return “this and obj are of the same class, and this and obj have the same x and y fields” */
    @Override
    public boolean equals(Object obj) {
        How can we tell whether this and obj are of the same class?
    }
}

function equals() requires casting
But, use of explicit down-casts can indicate bad design

DON'T: DO:
if (…) x.do()
    do something with (C1) x
else if (…) ... where do() is overridden in
    do something with (C2) x
classes C1, C2, C3
else if(…) do something with (C3) x

if (s[k] instanceof Circle) {
    Circle cir= Circle[s[k];
}