Overview ref in JavaHyperText

- Quick look at arrays \[\text{array}\]
- Casting among classes \[\text{cast}\]
- Operator \[\text{instanceof}\]
- Function \[\text{getClass}\]
- Function equals
- compile-time reference rule

Homework. JavaHyperText while-loop for-loop

```java
while (<?> expr) { … } // syntax
for (int k=0; k<200; k=k+1) { … } // example
```

Announcements

- Search Piazza for your question (before posting)!
- Partner-finding event:
  - Tuesday, September 12 at 5:30pm
  - Phillips 203
  - There will be snacks!

Before Next Lecture...

- Follow the tutorial on \textit{abstract classes and interfaces}, and watch the videos.

Classes we work with today

Work with a class \texttt{Animal} and subclasses like \texttt{Cat} and \texttt{Dog}

Put components common to animals in \texttt{Animal}

```java
class hierarchy:
  Object
  Animal
    Dog
    Cat

<table>
<thead>
<tr>
<th>a0</th>
<th>age</th>
<th>Animal</th>
<th>isOlder(Animal)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a1</th>
<th>age</th>
<th>Animal</th>
<th>isOlder(Animal)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cat</th>
<th>getNoise()</th>
<th>toString()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(…);</td>
<td>()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dog</th>
<th>getNoise()</th>
<th>toString()</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(…);</td>
<td>()</td>
</tr>
</tbody>
</table>

Object partition is there but not shown

```java
Animal[] v = new Animal[3];
```

Create array of 3 elements

Assign and refer to elements as usual:

```java
v[0] = new Animal(...);
...
a = v[0].getAge();
```

Sometimes use horizontal picture of an array:

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>null</td>
<td>null</td>
</tr>
</tbody>
</table>
```
Which function is called by `v[0].toString()`?

(remember, the hidden object partition contains `toString()`.)

Bottom-up or overriding rule says function `toString()` in `Cat` partition

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

`a.getPurrs()` is obviously illegal. The compiler will give you an error.

When checking legality of a call like `a.getPurrs()` since the type of `a` is Animal, method `getPurrs()` must be declared in Animal or one of its superclasses.

Rule for determining legality of method call

Rule: `c.m(…)` is legal and the program will compile ONLY if method `m` is declared in `C` or one of its superclasses. (JavaHyperText entry: compile-time reference rule.)

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

The same object `a0`, from the viewpoint of a Cat variable and an Animal variable

Rule: `c.m(…)` is legal and the program will compile ONLY if method `m` is declared in `C` or one of its superclasses. (JavaHyperText entry: compile-time reference rule.)

Consequences of a class type

Animal[] `v`; declaration of `v`. Also means that each variable `v[k]` is of type Animal

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

The type is part of the syntax/grammar of the language. Known at compile time.

A variable’s type:
* Restricts what values it can contain.
* Determines which methods are legal to call on it.
Another example

Type of v[0]: Animal

Should this call be allowed?
Should program compile?

v[0].getPurrs()

0 1 2

a0 null a1

getNoise() toString()

getPurrs()

age

isOlder(Animal)

Animal

Cat

Dog

View of object based on the type

Each element v[k] is of type Animal.
From v[k], see only what is in partition Animal and partitions above it.

Components are in lower partitions, but can’t see them

age

isOlder(Animal)

Animal

Cat

Dog

Casting objects

You know about casts like:

( int ) ( 5.0 / 7.5 )
( double ) 6
double d = 5; // automatic cast

You can also use casts with class types:

Animal h = new Cat( "N" , 5 );
Cat c = ( Cat ) h;

A class cast doesn’t change the object. It just changes the perspective on how it is viewed!

Explicit casts: unary prefix operators

Rule: At run time, 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 an exception

(Cat) c
(Object) c
(Animal) (Animal) (Cat) (Object) c

These casts don’t take any time. The object does not change. It’s a change of perception.

Implicit upward cast

public class Animal {
   /** = "this Animal is older than h" */
   public boolean isOlder(Animal h) {
      return age > h.age;
   }
}

c . isOlder( d )

Variable h is created. a1 is cast up to class Animal and stored in h
Upward casts done automatically when needed

h a1 Animal c a0 Cat d a1 Dog

getNoise() toString()

getPurrs()

getPurrs() not in class Animal or Object. Calls are illegal, program does not compile:

v[0].getPurrs() v[k].getPurrs()

Components are in lower partitions, but can’t see them

age

isOlder(Animal)

Animal

Cat

Dog

Example

public class Animal {
   /** = "this Animal is older than h" */
   public boolean isOlder(Animal h) {
      return age > h.age;
   }
}

Type of h is Animal. Syntactic property.

Determines at compile-time what components can be used: those available in Animal

If a method call is legal, the overriding rule determines which implementation is called

h a1 Animal
Components used from h

public class Animal {
    /** = "this is older than h" */
    public boolean isOlder(Animal h) {
        return age > h.age;
    }  

    String getPurrs();
    String getNoise();
    String toString();
}

Method getClass, explicit down cast

public class Cat extends Animal {
    private int purrs;
    /** return true iff ob is a Cat and its * fields have same values as this */
    public boolean equals(Object ob) {
        if (ob.getClass() != getClass()) return false;
        if (!super.equals(ob)) return false;
        Cat c = (Cat) ob;  // downward cast
        return purrs == c.getPurrs();
    }

    @Override
    public String getPurrs() {
        return super.getPurrs();
    }

    @Override
    public String toString() {
        return super.toString();
    }
}

Operator instanceof

// Both are true.
if (a0 instanceof Cat) ...
if (a0 instanceof Animal) ...

// Only the first is true.
if (a0.getClass() == Cat.class) ...
if (a0.getClass() == Animal.class) ...

true if <object> has a partition for <class-name>

Explicit downward cast

public class Cat extends Animal {
    private int purrs;
    /** return true iff ob is a Cat and its * fields have same values as this */
    public boolean equals(Object ob) {
        if (ob.getClass() != getClass()) return false;
        if (!super.equals(ob)) return false;
        Cat c = (Cat) ob;  // downward cast
        return purrs == c.getPurrs();
    }

    @Override
    public String getPurrs() {
        return super.getPurrs();
    }

    @Override
    public String toString() {
        return super.toString();
    }
}

A complete implementation of equals

public class Cat extends Animal {
    private int purrs;
    /** return true iff ob is a Cat and its * fields have same values as this */
    public boolean equals(Object ob) {
        if (ob == null || ob.getClass() != getClass()) return false;
        if (!super.equals(ob)) return false;
        Cat c = (Cat) ob;  // downward cast
        return purrs == c.getPurrs();
    }

    @Override
    public String getPurrs() {
        return super.getPurrs();
    }

    @Override
    public String toString() {
        return super.toString();
    }
}

Opinions about casting

- Use of instanceof and downcasts can indicate bad design
  DON'T:
  if [x instanceof C1] ...
  do thing with (C1) x
  else if [x instanceof C2] ...
  do thing with (C2) x
  ...
  where do is overridden in the classes C1, C2, C3

- But how do I implement equals()??
  That requires casting!