A2 is due Sunday

Everyone should get 100/100 since we gave you all the test cases you need.

Please look at the pinned Piazza note “Assignment A2” for information that is not in the handout and answers to questions.

Before Next Lecture...

Follow the tutorial on abstract classes and interfaces, and watch <13 minutes of videos.

Abstract classes and interfaces

This will prepare you for Thursday's lecture.

Classes we work with today

```
class Animal
subclasses Cat and Dog
Put components common to animals in Animal
```

```
cat pet1= new Cat(5);  
dog pet2= new Dog(6);  
```

Casting
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);
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 an exception

(Cat) c
(Object) c
(Cat) (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;
    }
}

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

// pet1 is cast up to class Animal and stored in h

h.toString() OK — it’s in class Object partition
h.isOlder(...) OK — it’s in Animal partition
h.purr() ILLEGAL — not in Animal partition or Object partition

Components used from h

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

h.toString() OK — it’s in class Object partition
h.isOlder(...) OK — it’s in Animal partition
h.purr() ILLEGAL — not in Animal partition or Object partition
Which toString() gets called?
See slide 18.

Compile-time reference rule

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

Animal pet1 = new Animal(5);

pet1.purr();

obviously illegal
The compiler will give you an error.

Checking the legality of pet1.purr(...):
Since pet1 is an Animal, purr must be declared in Animal or one of its superclasses.

From an Animal variable, can use only methods available in class Animal
13/09/2018

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

Animal pet1 = new Cat(5);
pet1.purr();

Still illegal!
The compiler still gives you an error.

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

Checking the legality of pet1.purr(…):
Since pet1 is an Animal, purr must be declared in Animal or one of its superclasses.

Why would we ever do this?

- Why would a variable of type Animal ever not have just an Animal in it?
- This is one of the beautiful things about OO programming!

1. We want to use an Animal method (seen)
2. We want to keep a list of all our pets
   - Create an array of type Animal!

Consequences of a class type

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.
Example: Point Class

```java
public class Point {
    public int x;
    public int y;

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

How Object defines equals(x)

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

Can define equals for your own class!

Can I define it any way I like?

https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-java.lang.Object-

Java spec says:
- Reflexive
- Symmetric
- Transitive

(click on the link to see what these are)

How do we define equality for a Point?

```java
/** return "obj is a Point and
 * obj and this have the same x and y fields" */
@Override
public boolean equals(Object obj) { // why Object?
    // how can we access the x y fields
    // if this is an Object?
    return false;
}
```

Use operator instanceof

```java
ob instanceof C
true iff ob has a partition named C
```

<table>
<thead>
<tr>
<th>ob</th>
<th>instanceof C</th>
<th>instanceof Animal</th>
<th>instanceof Cat</th>
<th>instanceof JFrame</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>h</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
</tbody>
</table>
How do we define equality for a Point?

```java
/** return "obj is a Point and
 * obj and this have the same x and y fields" */

@Override
public boolean equals(Object obj) {
    if (!((obj instanceof Point))
        return false;
    Point p = (Point)obj;
    return (x == p.x && y == p.y);
}
```

Opinions about casting

Use of instanceof and down-casts 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
else if (x instanceof C3)
    do thing with (C3) x

**DO:**
... where do is overridden in the
classes C1, C2, C3

But how do I implement equals()

That requires casting!

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) {
        // how to check that objects are of the
        // same class??
    }
}
```

Use function getClass

```
h.getClass()
```

Let Cat be the lowest partition of object h
Then
```
h.getClass() == Cat.class
```
```
h.getClass() != Animal.class
```

Equals in Animal

```
if (obj == null || getClass() != obj.getClass())
    return false;
Animal an = (Animal) obj; // cast obj to Animal!!!!
return age == an.age; // downcast needed to reference age
```