CS 1110
Prelim II: Review Session

Introduction

- My name: Bruno Abrahao
  - We have three additional TA's in the room to help you individually
    - Shuang
    - Nam
    - Yookyung
  - You're welcome to ask them questions at any time

Exam Info

- Prelim II: 7:30–9:00PM, Thursday, 12 March, Uris Hall G01
- Look at the previous Prelims
- Arrive early! Helps reducing stress
- Grades released the same evening (morning…)

Regrade Requests

- Releasing grades quickly is good for you — exams serve two purposes:
  - Give feedback to student and teacher
  - Give grades

That's one reason we grade ~180 exams so quickly

Review session

- Let's make this interactive
  - More fun
- Presentation is at slower pace than a regular lecture
- Ask questions
  - All questions are smart ones

What's in the exam?

- Everything you needed to know for Prelim I
- Vector / String class, functions
- Writing functions
- Recursive Functions
- apparent/real classes, casting, operator **instanceof**, function equals
- Abstract classes and methods

What's in the exam?

- Everything you needed to know for Prelim I
- Vector / String class, functions
- Writing functions
- Recursive Functions
- apparent/real classes, casting, operator **instanceof**, function equals
- Abstract classes and methods

I'm gonna assume you can do this with your eyes closed by now

What's in the exam?

- Everything you needed to know for Prelim I
- Vector / String class, functions
- Writing functions
- Recursive Functions
- apparent/real classes, casting, operator **instanceof**, function equals
- Abstract classes and methods

(Fall'07) Question 1 (15 points). Write the body of the following function recursively.

```
/** = n, but with its digits reversed.
      Precondition: n >= 0.
      e.g. n = 135720, value is "027531".
      e.g. n = 12345, value is "54321".
      e.g. n = 7, value is "7".
      e.g. n = 0, value is "0".*/
public static String rev(int n) {



}
```

returns a String

Recursive Function 4 Principles

- 1. Write the precise specification


```
/** = n, but with its digits reversed.
      Precondition: n >= 0.
   e.g. n = 135720, value is "027531".
   e.g. n = 12345, value is "54321".
   e.g. n = 7, value is "7".
   e.g. n = 0, value is "0".*/
public static String rev(int n) {
   // base case:
   //{n has only one digit}


   // recursive case:
   // {n has at least two digits}

}
```


Recursive Function 4 Principles

- 1. Write the precise specification
- 2. Base Case

```
/** = n, but with its digits reversed.
      Precondition: n >= 0.
   e.g. n = 135720, value is "027531".
   e.g. n = 12345, value is "54321".
   e.g. n = 7, value is "7".
   e.g. n = 0, value is "0".*/
public static String rev(int n) {
   // base case:
   //{n has only one digit}
   if (n < 10)

   // recursive case:
   // {n has at least two digits}

}
```

13

Let's review some type issues

What is the type of?

- `42`
- `"" + 42;`
- `'a' + 'b'`
- `'b' + "anana"`
- `'b' + 'a' + "nana"`
- `'b' + ('a' + "nana")`
- `"" + 'b' + 'a' + "nana"`

14

```
/** = n, but with its digits reversed.
      Precondition: n >= 0.
   e.g. n = 135720, value is "027531".
   e.g. n = 12345, value is "54321".
   e.g. n = 7, value is "7".
   e.g. n = 0, value is "0".*/
public static String rev(int n) {
   if (n < 10)                    base case:
      return "" + n;              n has 1 digit

   // recursive case:
   // {n has at least two digits}

}
```

15

Recursive Function 4 Principles

- 1. Write the precise specification
- 2. Base Case
- 3. Progress
  - Recursive call, the argument is "smaller than" the parameter. Ensures base case will be reached (which terminates the recursion)
- 4. Recursive case

16

```
/** = n, but with its digits reversed.
      Precondition: n >= 0.
   e.g. n = 135720, value is "027531".
   e.g. n = 12345, value is "54321".
   e.g. n = 7, value is "7".
   e.g. n = 0, value is "0".*/
public static String rev(int n) {
   if (n < 10)                    base case:
      return "" + n;              n has 1 digit

   // n has at least 2 digits     recursive case:
   return (n%10) + rev(n/10);

}
```

17

```
/** = the reverse of s.*/
public static String rev(String s) {
   if (s.length() <= 1)           base case
      return s;

   // { s has at least two chars }
   int k= s.length()-1;
   return s.charAt(k) +           recursive case
          rev(s.substring(1,k)) +
          s.charAt(0);
```

Do this one using this idea:
To reverse a string that contains at least 2 chars, switch first and last ones and reverse the middle.

18

What's in the exam?

- Everything you needed to know for Prelim I
- Vector / String class, functions
- Writing functions
- Recursive Functions
- apparent/real classes, casting, operator **instanceof**, function equals
- Abstract classes and methods

19

Administrivia

- Please remember to fill out your TA evals.
  - Open untill **Friday 13**
  - Good for **you**, good for **us**
  - You can give feedback to any TA (not only your lab instructor)
    - A TA who taught you something
    - A TA that inspired you
    - A TA who you think needs to improve some aspect
    - …

20

CS1110 Flix

21

```
public class Movie {
  private String title; // title of movie
  private int length; // length in minutes
  /** Constructor: document with title t
      and len minutes long */
  public Movie(String t, int len) {
    title= t; length= len;
  }
  /** = title of this Movie */
  public String getTitle()
    { return title; }
  /** = length of document, in minutes */
  public int getLength()
    { return length; }
  /** = the popularity:
      shorter means more popular */
  public int popularity()
    { return 240 - length; }
}
```

```
public class Trailer extends Movie {
  /** Constructor: a trailer of movie t.
    Trailers are 1 minute long*/
  public Trailer(String t)
    {super(t, 1);}
}
```

```
public class Documentary extends Movie {
  private String topic; // …

  /** Constructor: instance with title t,
      length n, and topic p */
  public Documentary(String t, int n,
                     String p) {
    super(t, n);
    topic= p;
  }

  /** = "Documentary" */
  public String DocumentaryType()
    { return "Documentary"; }

  /** = popularity of this instance */
  public int popularity()
    { return 200 - getLength(); }
}
```

```
public class Short extends Documentary {
  /** Constructor: instance with title t,
    length n, and topic p */
  public Short(String t, int n, String p)
    { super(t, n, p); }

  /** displays acknowledgement */
  public String showAck()
    {return "We thank our director";}

  /** = "Short Doc" */
  public String DocumentaryType()
    { return "Short Doc"; }
}
```

22

(Fall'05) **Question 4 (30 points)** For each pair of statements below, write the value of d after execution. If the statements lead to an error, write "BAD" and briefly explain the error. (The question continues on the next page.)

```
Documentary e=
        new Short("Man on Wire", 5, "Bio");
boolean d=
       "Short Doc" .equals(e.DocumentaryType());
```

23

(Fall'05) **Question 4 (30 points)** For each pair of statements below, write the value of d after execution. If the statements lead to an error, write "BAD" and briefly explain the error. (The question continues on the next page.)

```
Documentary e=
        new Short("Man on Wire", 5, "Bio");
boolean d=
       "Short Doc" .equals(e.DocumentaryType());
```

True. method equals here is from the string object

24

```
2.
Movie c=
    new Documentary(null, 3, "Carter Peace Center");

int d= c.popularity();
```

25

```
public class Movie {
  private String title; // title of movie
  private int length; // length in minutes
  /** Constructor: document with title t
      and len minutes long */
  public Movie(String t, int len) {
    title= t; length= len;
  }
  /** = title of this Movie */
  public String getTitle()
    { return title; }
  /** = length of document, in minutes */
  public int getLength()
    { return length; }
  /** = the popularity:
      shorter means more popular */
  public int popularity()
    { return 240 - length; }
}
```

```
public class Trailer extends Movie {
  /** Constructor: a trailer of movie t.
    Trailers are 1 minute long*/
  public Trailer(String t)
    {super(t, 1);}
}
```

```
public class Documentary extends Movie {
  private String topic; // …

  /** Constructor: instance with title t,
      length n, and topic p */
  public Documentary(String t, int n,
                     String p) {
    super(t, n);
    topic= p;
  }

  /** = "Documentary" */
  public String DocumentaryType()
    { return "Documentary"; }

  /** = popularity of this instance */
  public int popularity()
    { return 200 - getLength(); }
}
```

```
public class Short extends Documentary {
  /** Constructor: instance with title t,
    length n, and topic p */
  public Short(String t, int n, String p)
    { super(t, n, p); }

  /** displays acknowledgement */
  public String showAck()
    {return "We thank our director";}

  /** = "Short Doc" */
  public String DocumentaryType()
    { return "Short Doc"; }
}
```

26

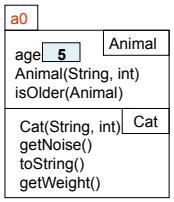
QUESTION: Which method is called by Animal t= new Cat("A",5); t.toString() ?

A. the one in the hidden partition for Object of a0

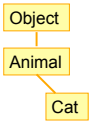
B. the one in partition Animal of a0

C. the one in partition Cat of a0

D. None of these

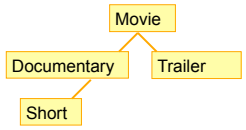


the class hierarchy:



27

```
2.
Movie c=
    new Documentary(null, 3, "Carter Peace Center");
int d= c.popularity();
```



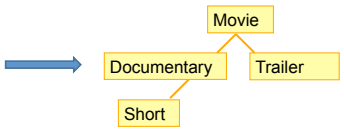
- What is the apparent class?
- **Answer: 197.** method popularity of class Documentary is called

28

```
3.
Short b= (Short)(new Documentary("", 2, "WMD"));
int d= b.DocumentaryType().length();
```

29

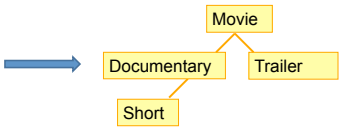
```
3.
Short b= (Short)(new Documentary("", 2, "WMD"));
int d= b.DocumentaryType().length();
```



- From documentary, can go (cast) up and back down to documentary.
- Think what would happen for the call b.showAck()

30

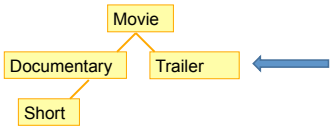
```
3.
Short b= (Short)(new Documentary("", 2, "WMD"));
int d= b.DocumentaryType().length();
```



- From documentary, can go (cast) up and back down to documentary.
- Think what would happen for the call b.showAck().
- Answer: BAD

31

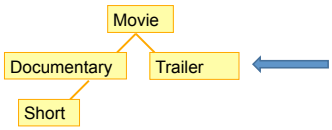
```
4.
Movie a= (Movie)(new Trailer("Harry Potter"));
int d= a.popularity();
```



- **The cast is legal!**
- **Which popularity() method is called?**

32

```
4.
Movie a= (Movie)(new Trailer("Harry Potter"));
int d= a.popularity();
```



- **The cast is legal!**
- Method popularity() from Movie is called (inherited by Trailer)
- **Answer: 239**

33

```
5.
Movie f= new Short("War", 1, "Vietnam");
char d= f.DocumentaryType().charAt(1);
```

The methods that can be called are determined by the apparent type:

Only components in the apparent class (and above)!!!

34

```
5.
Movie f= new Short("War", 1, "Vietnam");
char d= f.DocumentaryType().charAt(1);
```

The methods that can be called are determined by the apparent type:

Only components in the apparent class (and above)!!!

f.DocumentaryType() is illegal. Syntax error.

Answer: BAD

35

Recap: equals(Object ob)

- In class Object
  - b.equals(d) is the same as b == d
    - Unless b == null (why?)
- Most of the time, we want to use *equals* to compare fields. We need to override this method for this purpose

36

(Fall'05) Question 4 (24 points). (a) Write an instance method equals(Object obj) for class Documentary

```
public class Documentary extends Movie {
/** = "obj is a Documentary with the same values
      in its fields as this Documentary" */
public boolean equals(Object obj) {




}
```

37

```
public class Documentary extends Movie {
/** = "obj is a Documentary with the same values
      in its fields as this Documentary" */
public boolean equals(Object obj) {

  if (!(obj instanceof Documentary) {

  }




}
```

38

```
public class Documentary extends Movie {
/** = "obj is a Documentary with the same values
      in its fields as this Documentary" */
public boolean equals(Object obj) {

  if (!(obj instanceof Documentary) {
      return false;
  }




}
```

39

```
public class Documentary extends Movie {
/** = "obj is a Documentary with the same values
      in its fields as this Documentary" */
public boolean equals(Object obj) {

  if (!(obj instanceof Documentary) {
      return false;
  }
  Documentary docObj= (Documentary)obj;

          Don't forget to cast.
          This is a legal cast. (Why?)

}
```

40

```
public class Documentary extends Movie {
/** = "obj is a Documentary with the same values
      in its fields as this Documentary" */
public boolean equals(Object obj) {

  if (!(obj instanceof Documentary) {
      return false;
  }
  Documentary docObj= (Documentary)obj;
  return
    getTitle().equals(docObj.getTitle()) &&
    getLength() == docObj.getLength() &&
    topic.equals(docObj.topic);
```

41

What's in the exam?

- Everything you needed to know for Prelim I
- Vector / String class, functions
- Writing functions
- Recursive Functions
- apparent/real classes, casting, operator **instanceof**, function equals
- Abstract classes and methods

42

Let's capture the essence of animals

```
/** representation of an animal */
public class Animal {
   private int birthDate; // animal's birth date
   private String predator; // predator of this animal
   private String prey; // class of animals this hunts
   …
   // move the animal to direction…
   public void move(…){
     …
   }
   // make the animal eat…
   public void eat (…){
     …
   }
   …
}
```

43

Problems

- Animal is an abstract concept
  - Creating an abstract animal doesn't make sense in the real world
  - Dogs, cats, snakes, birds, lizards, all of which are animals, must have a way to eat so as to get energy to move
- However…
  - Class Animal allows us to create a UFA (unidentified flying animal), i.e. instance of Animal
  - If we extend the class to create a real animal, nothing prevent us from creating a horse that doesn't move or eat.

44

Solutions

- How to prevent one from creating a UFA?
  - Make class Animal abstract
    - **Class cannot be instantiated**
  - How? Put in keyword **abstract**
- How to prevent creation paralyzed dogs or starving sharks?
  - Make the methods move and eat abstract
    - **Method must be overridden**
  - How? Put in keyword **abstract** and replace the body with ";"

45

Making things abstract

```
/** representation of an animal */
public abstract class Animal{
   private int birthDate; // birth date
   private String predator; // animal's predator
   private String prey; // What animal hunts
   …
   // Move the animal move in direction  …
   public abstract void move(…);

   // Make the animal eat…
   public abstract void eat (…);
}
```

46

 Good Luck! 

47