CS 1110
Prelim II: Review Session

Exam Info

Prelim II: 7:30–9:00PM, Tuesday, 8 November,

Baker Lab 200, 219, 119

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 ~290 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
- Loops: for, while
- 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
- Loops: for, while
- 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
- Loops: for, while
- 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}

}
```

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"`


```
/** = 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)
      return "" + n;

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

}
```

base case: n has 1 digit

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


```
/** = 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)
      return "" + n;

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

}
```

base case: n has 1 digit

recursive case:


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

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

base case

recursive case

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.


```
/** reverses a given array of ints*/
public static void rev(int[] a) {

}
```

A word on arrays

An array is a container object that holds a fixed number of values of a single type.

The length of an array is established when the array is created.

After creation, its length is fixed.

```
//declares a variable that stores an int array
name
int[] a;

//creates an array of size 4
a = new int[4];

//creates and initializes an array of size 3
int [] b = new int[] {1,2,3};

//stores size of array in a variable
int size = b.length;

//swaps first and last elements in array b
int temp = b[0];
b[0] = b[1]
b[1] = temp;
```

```
/** reverses a given array of ints*/
public static void rev(int[] a) {

}

/** reverses h..k of a given array of ints */
private static void rev_in_range(int[] a, int
h,int k){




        }
```

```
/** reverses a given array of ints*/
public static void rev(int[] a) {
   rev_in_range(a,0,a.length-1);
}

/** reverses h..k of a given array of ints */
private static void rev_in_range(int[] a, int
h,int k){
   //Base case: k-h is 0 or -1
   if(k-h <= 0) return;
   //Recursive case

   }
```

```
/** reverses a given array of ints*/
public static void rev(int[] a) {
   rev_in_range(a,0,a.length-1);
}

/** reverses h..k of a given array of ints */
private static void rev_in_range(int[] a, int
h,int k){
   //Base case: k-h is 0 or -1
   if(k-h <= 0) return;
   //Recursive case
   int temp = a[h];
   a[h] = a[k];
   a[k] = temp;
   rev_in_range(a,h+1,k-1);
   }
```

```
/** reverses a given array of ints*/
public static void rev(int[] a) {
   rev_in_range(a,0,a.length-1);
}

/** reverses h..k of a given array of ints */
private static void rev_in_range(int[] a, int
h,int k){
   //Base case: k-h is 0 or -1
   if(k-h <= 0) return;
   //Recursive case
   int temp = a[h];
   a[h] = a[k];
   a[k] = temp;
   rev_in_range(a,h+1,k-1);
   }
```

Something horribly wrong here...

```
/** reverses a given array of ints*/
public static void rev(int[] a) {
   rev_in_range(a,0,a.length-1);
}

/** reverses h..k of a given array of ints */
private static void rev_in_range(int[] a, int
h,int k){
   //Base case: k-h is 0 or -1
   if(k-h <= 0) return;
   //Recursive case
   int temp = a[h];
   a[h] = a[k];
   a[k] = temp;
   rev_in_range(a,h+1,k-1);
   }
```

Something horribly wrong here...

Yes, bad indentation *is* horrible

Loops Part

For loops

We want to write a loop that calculates the sum of squares of the elements of an array `v` of `int`s.

1) Range of integers to be processed.
2) Write postcondition.
3) Write loop.
4) Write Invariant.
5) Write Initialization.
6) Process int in the range.

Loop Invariant

Invariant?

```
x = 0;
for (int i = 1; i <= 100; i = i+1)
    x = x + i;
```

Loop Invariant

- Invariant?
- **Inv: x = sum of integers in the range 1..(i-1)**
- x = 0;
- for (int i = 1; i <= 100; i = i+1)
  x = x + i;

Postcondition

- Invariant?
- **Inv: x = sum of integers in the range 1..(i-1)**
- Postcondition?

- x = 0;
- for (int i = 1; i <= 100; i = i+1)
  x = x + i;

Postcondition

- Invariant?
- **Inv: x = sum of integers in the range 1..(i-1)**
- Postcondition?
- **Post: x = sum of integers in the range 1..100**
- x = 0;
- for (int i = 1; i <= 100; i = i+1)
  x = x + i;

For loops

We want to write a loop that calculates the sum of squares of the elements of an array of `int`s.

```
// v is an array of ints.
// range: 0..v.length-1
int x = 0;
```

For loops

We want to write a loop that calculates the sum of squares of the elements of an array of `int`s.

```
// v is an array of ints.
// range: 0..v.length-1
int x = 0;




// postcondition:
// x = sum of squares of all the elements of v
```

For loops

We want to write a loop that calculates the sum of squares of the elements of an array of `int`s.

```
// v is an array of ints.
// range: 0..v.length-1
int x = 0;
// inv: x = sum of squares of v[0..i-1]
for (       ;           ;      ) {
    // process i

}
// postcondition:
// x = sum of squares of all the elements of v
```

For loops

We want to write a loop that calculates the sum of squares of the elements of an array of `int`s.

```
// v is an array of ints.
// range: 0..v.length-1
int x = 0;
// inv: x = sum of squares of v[0..i-1]
for (int i=0; i<v.length; i=i+1) {
    // process i

}
// postcondition:
// x = sum of squares of all the elements of v
```

For loops

We want to write a loop that calculates the sum of squares of the elements of an array of `int`s.

```
// v is an array of ints.
// range: 0..v.length-1
int x = 0;
// inv: x = sum of squares of v[0..i-1]
for (int i=0; i<v.length; i=i+1) {
    // process i
    x = x + v[i] * v[i];
}
// postcondition:
// x = sum of squares of all the elements of v
```

While loops

We are given a `Vector v` of `Integer`s and a threshold `t (int)`. We replace every value in the vector by 0 if it is <= `t;` by 1 otherwise.

```
// precondition: v is a Vector of Integers
// invariant:




// postcondition:
// every value in v has been replaced by 0
// if it was originally <= t, by 1 otherwise.
```

While loops

We are given a `Vector v` of `Integer`s and a threshold `t (int)`. We replace every value in the vector by 0 if it is <= `t;` by 1 otherwise.

```
// precondition: v is a Vector of Integers
// invariant:
//   for every j in the range 0..i-1,
//   v[j]=0 if the value of v at 0 was initially <= t;
//   v[j]=1 otherwise.




// postcondition:
// every value in v has been replaced by 0
// if it was originally <= t, by 1 otherwise.
```

While loops

We are given a `Vector v` of `Integer`s and a threshold `t (int)`. We replace every value in the vector by 0 if it is <= `t;` by 1 otherwise.

```
// precondition: v is a Vector of Integers
// invariant:
//   for every j in the range 0..i-1,
//   v[j]=0 if the value of v at 0 was initially <= t;
//   v[j]=1 otherwise.
int i=0;
while (i < v.size()) {


    i = i+1;
}
// postcondition:
// every value in v has been replaced by 0
// if it was originally <= t, by 1 otherwise.
```

While loops

We are given a `Vector v` of `Integer`s and a threshold `t (int)`. We replace every value in the vector by 0 if it is <= `t;` by 1 otherwise.

```
// precondition: v is a Vector of Integers
// invariant:
//   for every j in the range 0..i-1,
//   v[j]=0 if the value of v at 0 was initially <= t;
//   v[j]=1 otherwise.
int i=0;
while (i < v.size()) {
    int x = (Integer) v.get(i);
    v.set(i,x <= t ? 0 : 1);
    i = i+1;
}
// postcondition:
// every value in v has been replaced by 0
// if it was originally <= t, by 1 otherwise.
```

What's in the exam?

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

Apparent type: appeared type of object
Animal a = new Cat();
Animal is the apparent type

Real type: real type of the object
Animal a = new Cat();
Cat is the real type

instanceof: operator. Test the class of an object
Animal a = new Cat();
(a instanceof Animal) == true
(a instanceof Cat) == true

```
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 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(); }
}
```

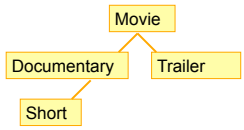
```
public class Trailer extends Movie {
  /** Constructor: a trailer of movie t.
    Trailers are 1 minute long*/
  public Trailer(String t)
    {super(t, 1);}
}
```

```
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"; }
}
```


Class Hierarchy



(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());
```


(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

2.

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

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



```
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 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(); }
}
```

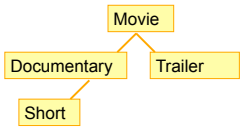
```
public class Trailer extends Movie {
  /** Constructor: a trailer of movie t.
    Trailers are 1 minute long*/
  public Trailer(String t)
    {super(t, 1);}
}
```

```
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"; }
}
```

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



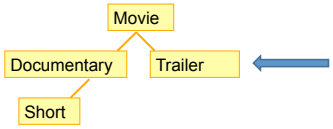
- What is the apparent class?
- **Answer: 197.** method popularity of class Documentary is called


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

java.lang.ClassCastException: From Documentary to Short

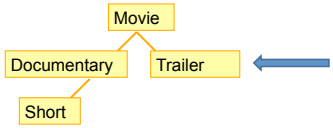
You don't know if a documentary is a short


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



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


```
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**


```
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)!!!


```
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

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

(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) {




}
```



```
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) {

  }



}
```



```
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;
  }



}
```



```
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?)

}
```



```
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);
```

What's in the exam?

Everything you needed to know for Prelim I

Vector / String class, functions

Writing functions

Recursive Functions

Loops: for, while

apparent/real classes, casting, operator **instanceof**, function equals

Abstract classes and methods

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 (…){
      …
  }
  …
}
```

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.

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 ";"

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 (…);
}
```


Good Luck!