

## CS 1110 Prelim I: Review Session

## Introduction

- My name: Bruno Abrahao
  - We have four TA's in the room to help you individually
    - Shuang Zhao
    - Nam Nguyen
    - Ankur Agarwal
    - Suyong Lee
  - You're welcome to ask them questions at any time

## Exam Info

- Prelim 1: 7:30–9:00PM, Thursday, 19 February, Baker Lab 200
- Look at the previous Prelims
- Arrive early! Helps reducing stress
- Grades released the same evening (morning...)

## Exam Info

- Prelim 1: 7:30–9:00PM, Thursday, 19 February, Baker Lab 200
- Look at the previous Prelims
- Arrive early! Helps reducing stress
- Grades released the same evening (morning...)



## What's in the exam?

- Definitions of terms and key concepts
- Execution of assignment statements
- Evaluation / Execution of "new" expressions
- Evaluation of method calls
- Execute sequence of statements
- String functions
- Writing class definitions

## We have a lot to "cover"



### What's in the exam?

- **Definitions of terms and key concepts**
- Execution of assignment statements
- Evaluation Execution of "new" expressions
- Evaluation of method calls
- String functions
- Writing class definitions

### Definitions

- Usually the first question of the exam
- Short answers
- Questions ask for definitions or something to be done
- Let's see 3 examples from Prelim I, spring'07.

### Definitions

- **(a) 5 pts.** What is an argument? A parameter?
- **(b) 5 pts.** What is a local variable? What is its scope?

### Definitions

- **(a) 5 pts.** What is an argument? A parameter?  
A parameter is a variable declared in the header of a method (within the parentheses). An argument is an expression that occurs in a call of a method.
- **(b) 5 pts.** What is a local variable? What is its scope?

### Definitions

- **(a) 5 pts.** What is an argument? A parameter?  
A parameter is a variable declared in the header of a method (within the parentheses). An argument is an expression that occurs in a call of a method.
- **(b) 5 pts.** What is a local variable? What is its scope?  
A local variable is a variable that is declared in the body of a method. Its scope begins at its declaration and continues until the end of the block in which it is declared.

### Definitions

- **(c) 5 pts.** Explain the three steps in evaluating a new-expression (e.g. `new Time(c,d)`). The previous sentence contains an example of a new-expression, but your answer should explain what *any* new-expression does and not simply the one in the example.

## Definitions

- 1. (c)
- For a new-expression `new C(...)`:
  - (1) Create a new object of class C;
  - (2) execute constructor call C(...);
  - (3) yield as value the name of the newly created object.
- This question asks us to explain, later we will be asked to execute!

## What's in the exam?

- Definitions of terms and key concepts
- Execution of assignment statements
- Evaluation / Execution of “new” expressions
- Evaluation of method calls
- String functions
- Writing class definitions

## Assignments

**Assignment statement.** p. 27 Execution of an assignment statement stores a value in a variable.

To execute the assignment `<var>= <expr>;`  
evaluate expression `<expr>` and store its value in variable `<var>`.

`x = x + 1;` Evaluate expression `x+1` and store its value in variable `x`.

```
v1= 5;           // v1: 5 v2: ?
v2= 3;           // v1: 5 v2: 3
v1= v1 + 1       // v1: 6 v2: 3
v1= v2 + v1 + 42 // v1: 51 v2: 3
```

15

## What's in the exam?

- Definitions of terms and key concepts
- Execution of assignment statements
- Evaluation / Execution of “new” expressions
- Evaluation of method calls
- String functions
- Writing class definitions

## Execute “new” statements

```
/** An instance represents a
  Student*/
public class Student {
  // the student's name
  private String name;
  // the student's netid
  private String netid;
  /** Constructor: a Person with
   name n and netid i */
  public Student(String n, String i)
  {name= n; netid= i;}
  /** set the Student's name to n */
  public void setName(String n)
  {name= n;}
  /** = "this Student and s have the
   same netid" */
  public boolean equals(Student p)
  {return netid.equals(p.netid);}
```

**Question 4 (20 points).** Below is a definition of class Student. Assume the following three assignment statements are executed:

```
Student p1= new Student("Bill", "bk12");
Student p2= new Student("Bill", "bk13");
Student p3= new Student("William", "bk12");
```

(a) What is the value of each of the following four expressions?

```
p1.equals(p2)
p1.equals(p3)
p1 == p2
p1 == p3
```

(b) Now consider these statements:

```
p1= new Student("Bill", "bk12");
p2= new Student("Bill", "bk13");
p3= p2;
p3.setName("Jack");
```

Below, first draw all three variables. Then execute the four statements —of course, draw any objects that are created during execution.

## A message from Profs. Gries and Lee

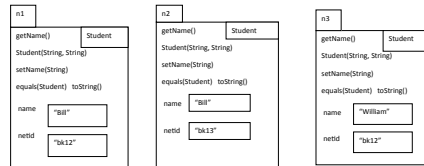
### A message from Profs. Gries and Lee

- You won't get these questions correct unless you draw the variables, objects, and execute their assignments!



### So let's draw them...

- Student p1= new Student("Bill", "bk12"); p1
- Student p2= new Student("Bill", "bk13"); p2
- Student p3= new Student("William", "bk12"); p3



- (a) What is the value of each of the following four expressions?
- p1.equals(p2)
  - p1.equals(p3)
  - p1 == p2
  - p1 == p3

- (a) What is the value of each of the following four expressions?
- p1.equals(p2) False
  - p1.equals(p3) True
  - p1 == p2 False
  - p1 == p3 False

- (b) Now consider these statements:

```
p3= p2;
p3.setName("Jack");
```



Below, first draw all three variables. Then execute the two statements.

- (b) Now consider these statements:

```
p3= p2;
p3.setName("Jack");
```



Below, first draw all three variables. Then execute the two statements.

**(b) Now consider these statements:**  
`p3 = p2;`  
`p3.setName("Jack");`

Below, first draw all three variables. Then execute the two statements.

### What's in the exam?

- Definitions of terms and key concepts
- Execution of assignment statements
- Evaluation / Execution of "new" expressions
- Evaluation of method calls
- String functions
- Writing class definitions

#### The frame (the box) for a method call

**Remember:** Every method is in a folder (object) or in a file-drawer.

Draw the parameters as variables.

number of the statement of method body to execute next. Helps you keep track of what statement to execute next. Start off with 1.

scope box contains the name of entity that contains the method — a file drawer or object.

To execute the call `x.setAmt(50);`

To execute the call `x.setAmt(50);`

1. Draw a frame for the call.

To execute the call `x.setAmt(50);`

1. Draw a frame for the call.

2. Assign the value of the argument to the parameter (in the frame).

To execute the call `x.setAmt(50);`

1. Draw a frame for the call.
2. Assign the value of the argument to the parameter (in the frame).
3. Execute the method body. (Look for variables in the frame; if not there, look in the place given by the scope box.)

31

To execute the call `x.setAmt(50);`

1. Draw a frame for the call.
2. Assign the value of the argument to the parameter (in the frame).
3. Execute the method body. (Look for variables in the frame; if not there, look in the place given by the scope box.)
4. Erase the frame for the call.

32

To execute the call `x.setAmt(50);`

1. Draw a frame for the call.
2. Assign the value of the argument to the parameter (in the frame).
3. Execute the method body. (Look for variables in the frame; if not there, look in the place given by the scope box.)
4. Erase the frame for the call. **But not in the exam! Leave it there for us to see!**

33

(d) 10 pts. Draw a frame for the call `m(2+3, 6)` of the following procedure `m`. We want to see what the frame for the call looks like after the argument values are assigned to the parameters but before the method body is executed.

```
public void m(int x, double y) {
    int z;
    z = x + y;
}
```

- ### What's in the exam?
- Definitions of terms and key concepts
  - Execution of assignment statements
  - Evaluation / Execution of "new" expressions
  - Evaluation of method calls
  - String functions
  - Writing class definitions

Return	Method	Purpose
char	<code>s.charAt(i)</code>	= the character at position <code>i</code> of <code>s</code>
int	<code>s.length()</code>	= the number of characters in <code>s</code>
int	<code>s.indexOf(n)</code>	= the index within <code>s</code> of the first occurrence of <code>String n</code> (-1 if none)
String	<code>s.substring(h,k)</code>	= a <code>String</code> consisting of characters in <code>s[h..k-1]</code> , i.e. <code>s[h], s[h+1], ..., s[k-1]</code>
String	<code>s.substring(h)</code>	= a <code>String</code> consisting of characters <code>s[h..s.length()-1]</code>

**Question 5: (20 pts)** Write function `fix`, which is specified below. You may use the following methods (you may not need them all). This might help you: when you break `String s` up into pieces, store the pieces in local variables and then use these pieces.

```
/** = Date s in a more suitable form.
Precondition: s contains a date in the form month/day/year, with each
part separated by "/". Examples are: 4/26/39 and 04/005/1939.
The output should be in the form year.month.day. Examples are:
39.26.4 and 1939.04.005.
Each of day, month, and year may be any length. They appear in
exactly the same form in the input and output; just their order and
the separator are changed. */
public static String fix(String s) {
```

```

/** = Date s in a more suitable form.
Precondition: s contains a date in the form month/day/year, with each
part separated by "/". Examples are: 4/26/39 and 04/005/1939.
The output should be in the form year.month.day. Examples are: 39.26.4
and 1939.04.005.
Each of day, month, and year may be any length. They appear in exactly
the same form in the input and output; just their order and the
separator are changed. */
public static String fix(String s) {
    int k = s.indexOf("/"); // index of first "/"
    String month = s.substring(0,k);
    String rest = s.substring(k+1);
    k = rest.indexOf("/"); // index in rest of the only "/"
    String day = rest.substring(0,k);
    String year = rest.substring(k+1);
    return year + "." + month + "." + day;
}

```

## What's in the exam?

- Definitions of terms and key concepts
- Execution of assignment statements
- Evaluation / Execution of "new" expressions
- Evaluation of method calls
- String functions
- Writing class definitions

```

/** An instance represents an
instrument */
public class MusicInstrument {
    // Member of Congress' name
    private String name = null;
    /** Constructor: an instrument with
name s */
    public MusicInstrument(String s) {
        name = s;
    }
    /** Constructor: an instrument with
name "" */
    public MusicInstrument() {
        name = "";
    }
    /** = sound this instrument makes */
    public String play() {
        return "music ";
    }
    /** = a repr of this instrument */
    public String toString() {
        return "Instrument: " + name;
    }
}

/** An instance represents a string
instrument with no name */
public class StringInstrument extends
MusicInstrument {
    /** number of strings on this
instrument */
    private int numStrings;
    /** Set the number of Strings on this
instrument to n
public void setNumber(int n) {
    numStrings = n;
}
    /** = sound this instrument makes */
    public String play() {
        return super.toString() +
numStrings + "trillings";
    }
}

```

### Question 2 (20 points) Write a class definition for a class PercussionInstrument that

- Is a subclass of MusicInstrument;
- Has suitable specifications on methods and definitions on fields;
- Has a field numDrums, which is the number of drums in this percussion instrument;
- Has a constructor with the name of the instrument and the number of drums as parameters;
- Overrides function play() to return the number of "druuums", similar to the way function play in class StringInstrument works.

1. Is a subclass of MusicInstrument;

```

/** An instance represents a percussion instrument */
public class PercussionInstrument extends MusicInstrument{
}

```

2. Has a field numDrums, which is the number of drums in this percussion instrument;

```

/** An instance represents a percussion instrument */
public class PercussionInstrument extends MusicInstrument{
    // number of drums in this instrument
    private int numDrums;
}

```

3. Has a constructor with the name of the instrument and the number of drums as parameters;

```

/** An instance represents a percussion instrument */
public class PercussionInstrument extends MusicInstrument{

    // number of drums in this instrument
    private int numDrums;

}
    
```

```

/** An instance represents an instrument */
public class MusicInstrument {
    // Member of Congress' name
    private String name= null;
    /** Constructor: an instrument with name s */
    public MusicInstrument(String s) {
        name= s;
    }
    /** Constructor: an instrument with name "" */
    public MusicInstrument() {
        name= "";
    }
    /** = sound this instrument makes*/
    public String play() {
        return "music ";
    }
    /** = a repr of this instrument */
    public String toString() {
        return "Instrument: " + name;
    }
}

/** An instance represents a string instrument with no name */
public class StringInstrument extends MusicInstrument {
    /** number of strings on this instrument */
    private int numStrings;
    /** Set the number of Strings on this instrument to n
    public void setNumber(int n) {
        numStrings= n
    }
    /** = sound this instrument makes */
    public String play() {
        return super.toString() + numStrings + "trillings";
    }
}
    
```

```

public class Executive extends Employee {
    private double bonus;
    /** Constructor: name n, year hired d, salary 50,000, bonus b */
    public Executive(String n, int d, double b) {
        super(n, d);
        bonus= b;
    }
}
    
```

Calling a superclass constructor from the subclass constructor  
 Sec. 4.1.3, page 147

a0

toString() ...	Object
salary 50,000	Employee
name "Gries" start 1969	Employee(String, int)
toString() getCompensation()	Employee(String, int)
bonus 10,000	Executive
Executive(String, int, double)	Executive
getBonus() getCompensation() toString()	Executive

The first (and only the first) statement in a constructor has to be a call to a constructor of the superclass.

Principle: Fill in superclass fields first.

45

3. Has a constructor with the name of the instrument and the number of drums as parameters;

```

/** An instance represents a percussion instrument */
public class PercussionInstrument extends MusicInstrument{

    // number of drums in this instrument
    private int numDrums;

    /** Constructor: an instance name s with n drums */
    public PercussionInstrument(String s, int n) {
        super(s);
        numDrums= n;
    }

}
    
```

4. Overrides function play() to return the number of "druuums", similar to the way function play in class StringInstrument works.

```

/** An instance represents a percussion instrument */
public class PercussionInstrument extends MusicInstrument{

    // number of drums in this instrument
    private int numDrums;

    /** Constructor: an instance name s with n drums */
    public PercussionInstrument(String s, int n) {
        super(s);
        numDrums= n;
    }

}
    
```

```

/** An instance represents an instrument */
public class MusicInstrument {
    // Member of Congress' name
    private String name= null;
    /** Constructor: an instrument with name s */
    public MusicInstrument(String s) {
        name= s;
    }
    /** Constructor: an instrument with name "" */
    public MusicInstrument() {
        name= "";
    }
    /** = sound this instrument makes*/
    public String play() {
        return "music ";
    }
    /** = a repr of this instrument */
    public String toString() {
        return "Instrument: " + name;
    }
}

/** An instance represents a string instrument with no name */
public class StringInstrument extends MusicInstrument {
    /** number of strings on this instrument */
    private int numStrings;
    /** Set the number of Strings on this instrument to n
    public void setNumber(int n) {
        numStrings= n
    }
    /** = sound this instrument makes */
    public String play() {
        return super.toString() + numStrings + "trillings";
    }
}
    
```



**Purpose of super and this** Sec. 4.1, pages 144-145  
**this** refers to the name of the object in which it appears.  
**super** is similar but refers only to components in the partitions above.

```

/** = String representation of this Employee */
public String toString() {
    return this.getName() + ", year " +
        getStart() + ", salary " + salary;
}
    
```

ok, but unnecessary

```

/** = toString value from superclass */
public String toStringUp() {
    return super.toString();
}
    
```

necessary

a0

```

equals(Object)
toString()
    
```

Object	
Employee	
name	"Gries"
salary	50,000.00
start	1969

```

getName()
setName(String n) {...}
toString()
toStringUp() {...}
    
```

```

/** An instance represents an instrument */
public class MusicInstrument {
    // Member of Congress' name
    private String name= null;
    /** Constructor: an instrument with name s */
    public MusicInstrument(String s) {
        name= s;
    }
    /** Constructor: an instrument with name "" */
    public MusicInstrument() {
        name= "";
    }
    /** = sound this instrument makes */
    public String play() {
        return "music ";
    }
    /** = a repr of this instrument */
    public String toString() {
        return "Instrument: " + name;
    }
}
    
```

```

/** An instance represents a string instrument with no name */
public class StringInstrument extends MusicInstrument {
    /** number of strings on this instrument */
    private int numStrings;
    /** Set the number of Strings on this instrument to n
    public void setNumber(int n) {
        numStrings= n
    }
    /** = sound this instrument makes */
    public String play() {
        return super.toString() +
            numStrings + "triiings";
    }
}
    
```

5. Overrides function play() to return the number of "druuums", similar to the way function play in class StringInstrument works.

```

/** An instance represents a percussion instrument */
public class PercussionInstrument extends MusicInstrument{
    // number of drums in this instrument
    private int numDrums;
    /** Constructor: an instance name s with n drums */
    public PercussionInstrument(String s, int n) {
        super(s);
        numDrums= n;
    }
    /** = sound this instrument makes */
    public String play() {
        return super.toString() + numDrums + "druuums";
    }
}
    
```

6. Has suitable specifications on methods and definitions on fields

```

/** An instance represents a percussion instrument */
public class PercussionInstrument extends MusicInstrument{
    // number of drums in this instrument
    private int numDrums;
    /** Constructor: an instance name s with n drums */
    public PercussionInstrument(String s, int n) {
        super(s);
        numDrums= n;
    }
    /** = sound this instrument makes */
    public String play() {
        return super.toString() + numDrums + "druuums";
    }
}
    
```

6. Has suitable specifications on methods and definitions on fields **Done!**

```

/** An instance represents a percussion instrument */
public class PercussionInstrument extends MusicInstrument{
    // number of drums in this instrument
    private int numDrums;
    /** Constructor: an instance name s with n drums */
    public PercussionInstrument(String s, int n) {
        super(s);
        numDrums= n;
    }
    /** = sound this instrument makes */
    public String play() {
        return super.toString() + numDrums + "druuums";
    }
}
    
```

## Summary

- Definitions of terms and key concepts
- Execution of assignment statements
- Evaluation / Execution of "new" expressions
- Evaluation of method calls
- String functions
- Writing class definitions

Question period

