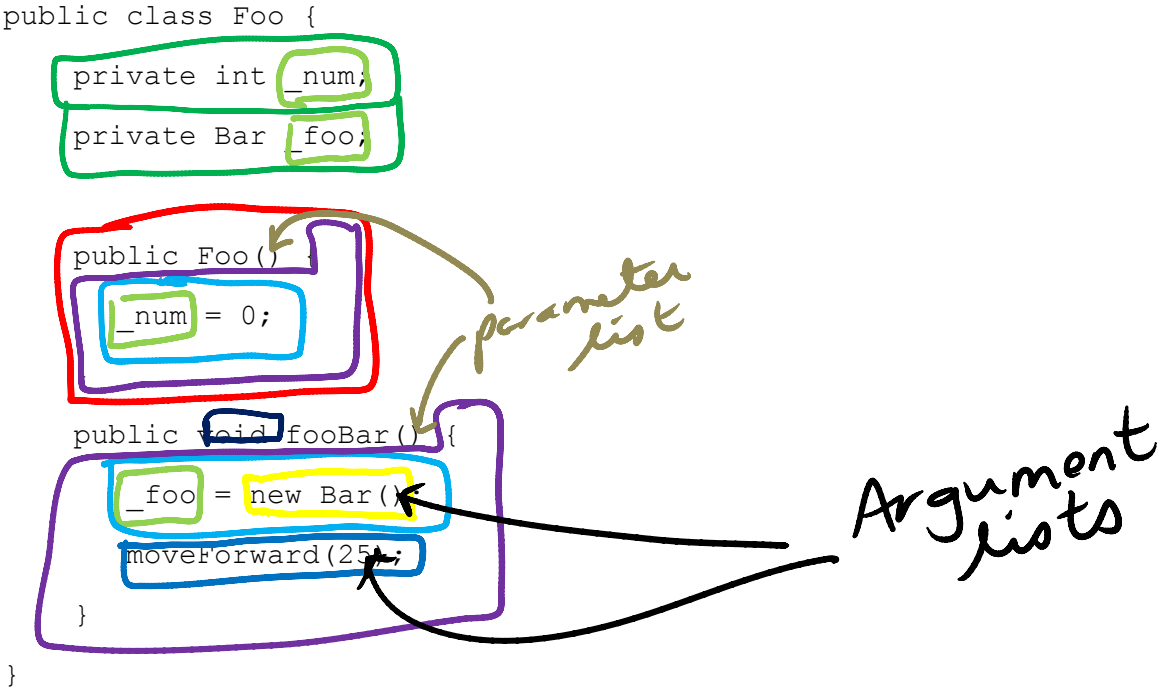

1. Use the class definition above to circle and identify the parts of code from the list given.

a) Constructor definition
b) Code that creates an object
c) Instance variable name
d) Instance variable declaration
e) Assignment statement
f) Method call
g) Method body
h) Return type of a method
i) Parameter list

2. Based on this method definition, answer parts a –d.

```
public School getSchool() {

}
```

a) Which of the following is the name of the method?
   - public
   - School
   - getSchool
   - ()
   - { }

b) Which of the following is the parameter list of the method?
   - public
   - School
   - getSchool
   - ()
   - { }

c) Which of the following is the body of the method?
- public
- School
- getSchool
- ()
- { }

d) Which of the following is the return type of the method?
- public
- School
- getSchool
- ()
- { }

3. Fill in the code for the following if statement so that the action given will happen 25% of the time.

```
if( Greenfoot.getRandomNumber(100) < 25 )
{
      shout("Yeah!");
}
```

4. Fill in the parameters to turn so that the number of degrees to turn will be a random number between 1 and 100.

turn((Greenfoot.getRandomNumber(100)+1))

5. What is the purpose of the constructor in code? When is a constructor called/executed?

The constructor sets up the initial state of the object. The constructor is called whenever we create an new object using the keyword new.

For questions 6 - 9, fill assume that the method calls will go in the space indicated in the code sample given.

```
public class Forrest extends World {

  public Forrest() {

      //Code for Questions 6-9 would be written here
      //Question 6 answer
      addObject(new Leaf(), 45, 36);

      //Question 7 answer
      addObject(new Leaf(), Greenfoot.getRandomNumber(getWidth()), Greenfoot.getRandomNumber(getHeight()));

      //Question 8 answer
      addObject(new Leaf(), getWidth(), getHeight());

      //Question 9 answer
      int count = 0;
      while(count < 5) {
         addObject(new Leaf(), Greenfoot.getRandomNumber(getWidth()), Greenfoot.getRandomNumber(getHeight()));
         count = count + 1;
      }
  }
}
```

6. Write the method call to add a Leaf to the world at location (45, 36).

7. Write the method call to add a Leaf to the world at a random location.

8. Write the method call to add a Leaf to the world at the lower right hand corner of the world.

9. Write the code that adds 5 Leaf objects to the world at random locations (using a loop).

The following questions are not related to the above example.

10. Write the code to create a Boy object.

```
new Boy()
```

11. Write the code to declare an instance variable of type Boy named _boy.

```
private Boy _ boy;
```

12. Write the code that assigns the value 45 to a variable named temp.

```
temp = 45;
```

13. Write the code that declares a variable whose type is an array that holds integers. The name of the variable should be nums.

```
int[] nums;
```

14. Write the code that inserts the even numbers from 0 to 14 into the array nums.

```
nums = {0, 2, 4, 6, 8, 10, 12, 14};
```

Other topics to study:

- if/else statements
- boolean logical operators (and, or, not)
- Loops
- How we made multiple keys for the piano out of one class definition (abstraction)

Actor Method Summary

void	**act**() The act method is called by the greenfoot framework to give objects a chance to perform some action.
protected void	**addedToWorld**(World world) This method is called by the Greenfoot system when the object has been inserted into the world.
GreenfootImage	**getImage**() Returns the image used to represent this Actor.
protected java.util.List	**getIntersectingObjects**(java.lang.Class cls) Return all the objects that intersect this object.
protected java.util.List	**getNeighbours**(int distance, boolean diagonal, java.lang.Class cls) Return the neighbours to this object within a given distance.
protected java.util.List	**getObjectsAtOffset**(int dx, int dy, java.lang.Class cls) Return all objects that intersect the given location (relative to this object's location).
protected java.util.List	**getObjectsInRange**(int radius, java.lang.Class cls) Return all objects within range 'radius' around this object.
protected Actor	**getOneIntersectingObject**(java.lang.Class cls) Return an object that intersects this object.
protected Actor	**getOneObjectAtOffset**(int dx, int dy, java.lang.Class cls) Return one object that is located at the specified cell (relative to this objects location).
int	**getRotation**() Return the current rotation of the object.
World	**getWorld**() Return the world that this object lives in.
int	**getX**() Return the x-coordinate of the object's current location.
int	**getY**() Return the y-coordinate of the object's current location.
protected boolean	**intersects**(Actor other) Check whether this object intersects with another given object.
void	**setImage**(GreenfootImage image) Set the image for this object to the specified image.
void	**setImage**(java.lang.String filename) Set an image for this object from an image file.
void	**setLocation**(int x, int y) Assign a new location for this object.
void	**setRotation**(int rotation) Set the rotation of the object.

World Method Summary

void	**act**() Act method for world.
void	**addObject**(Actor object, int x, int y) Add an Actor to the world.
GreenfootImage	**getBackground**() Return the world's background image.
int	**getCellSize**() Return the size of a cell (in pixels).
java.awt.Color	**getColorAt**(int x, int y) Return the color at the centre of the cell.
int	**getHeight**() Return the height of the world (in number of cells).
java.util.List	**getObjects**(java.lang.Class cls) Get all the objects in the world, or all the objects of a particular class.
java.util.List	**getObjectsAt**(int x, int y, java.lang.Class cls) Return all objects at a given cell.
int	**getWidth**() Return the width of the world (in number of cells).
int	**numberOfObjects**() Get the number of actors currently in the world.
void	**removeObject**(Actor object) Remove an object from the world.
void	**removeObjects**(java.util.Collection objects) Remove a list of objects from the world.
void	**repaint**() Repaints the world.
void	**setActOrder**(java.lang.Class... classes) Set the act order of objects in the world.
void	**setBackground**(GreenfootImage image) Set a background image for the world.
void	**setBackground**(java.lang.String filename) Set a background image for the world from an image file.
void	**setPaintOrder**(java.lang.Class... classes) Set the paint order of objects in the world.
void	**started**() This method is called by the Greenfoot system when the execution has started.
void	**stopped**() This method is called by the Greenfoot system when the execution has stopped.