SOLUTIONS

1. **Binary Search Trees** (10 points)

The Java `TreeMap` class is implemented using a Binary Search Tree. This class maintains the Binary Search Tree invariant in its implementation, storing the entries in the tree in order, sorted by the keys. Based on your understanding of BSTs, which of the following methods of this class should make use of the BST invariant?

(a) `boolean containsKey(Object key)`
Returns true if this map contains a mapping for the specified key.

☒ Yes ☐ No

(b) `boolean containsValue(Object value)`
Returns true if this map maps one or more keys to the specified value.

☐ Yes ☒ No

(c) `K firstKey()`
Returns the first (lowest) key currently in this map.

☒ Yes ☐ No

(d) `V get(Object key)`
Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.

☒ Yes ☐ No

(e) `int size()`
Returns the number of key-value mappings in this map.

☐ Yes ☒ No

2. **OCaml Queues**

Recall the linked list implementation of the `queue` type from HW 4. For reference, the definition of the `queue` type and basic operations appear in Appendix A (and in `queue.ml` in the OCaml Codio project).

The `contains1` function, shown below, correctly determines whether a given element is contained in a queue.

```
1 let contains1 (q : 'a queue) (x: 'a) : bool =
2   let rec loop qno =
3     begin match qno with
4       | None -> false
5       | Some n ->
6             n.v == x
7         || loop n.next
8     end
9 in loop q.head
```

(a) (2 points) What line of code can we fill in the blank below so that the following test case passes? (There may be more than one correct answer, and all options should be considered individually. Mark an 'x' next to **all appropriate responses**.)

```
;; run_test "contains1" (fun () ->
      let q = create () in
      let x = 5    in

      _____________________________
      contains1 q x)
```

☒ `enq 5 q;` ☒ `enq x q;` ☐ No possible line would work

☐ `enq 3 q;` ☐ `enq x x;` ☐ Leave it blank (already passes)

(b) (2 points) What line of code can we fill in the blank so that the following test case passes? (There may be more than one correct answer, and all options should be considered individually. Mark an 'x' next to **all appropriate responses.**)

```
;; run_test "contains1" (fun () ->
      let q = create () in
      let x = Some 3    in

      _____________________________
      contains1 q x)
```

☐ `enq (Some 3) q;` ☒ `enq x q;` ☐ No possible line would work

☐ `enq 3 q;` ☐ `enq x x;` ☐ Leave it blank (already passes)

Suppose that we modify line 6 of `contains` so that it reads `n.v = x`.

```
1 let contains2 (q : 'a queue) (x: 'a) : bool =
2   let rec loop qno =
3     begin match qno with
4       | None -> false
5       | Some n ->
6             n.v = x
7         || loop n.next
8     end
9 in loop q.head
```

(c) (2 points) What line of code can we fill in the blank so that the following test case passes? (There may be more than one correct answer, and all options should be considered individually. Mark an 'x' next to **all appropriate responses.**)

```
;; run_test "contains2" (fun () ->
      let q = create () in
      let x = Some 3    in

      _____________________________
      contains2 q x)
```

☒ `enq (Some 3) q;` ☒ `enq x q;` ☐ No possible line would work

☐ `enq 3 q;` ☐ `enq x x;` ☐ Leave it blank (already passes)

(d) (2 points) What line of code can we fill in the blank so that the following test case passes? (There may be more than one correct answer, and all options should be considered individually. Mark an 'x' next to **all appropriate responses.**)

```
;; run_test "contains2" (fun () ->
      let q1 = create () in
      let q2 = create () in

      _____________________________

      contains2 q1 q2)
```

☒ `enq q2 q1;` ☒ `enq (create ()) q1;` ☐ No possible line would work.

☐ `enq q1 q2;` ☐ `enq q2 q2;` ☐ Leave it blank. (Test already passes)

(e) (7 points) Now generalize the `contains1` and `contains2` functions by adding a higher-order function as an argument. Your new function version should be called `exists` and should have the type as shown below. It should be possible to use your `exists` function to define both versions.

For example, the definition of the `contains` function from part (a) using `exists` should be:

```
let contains1 (q:'a queue) (x:'a) = exists q (fun v -> v == x)
```

and the definition of the `contains` function from part (c) should be:

```
let contains2 (q:'a queue) (x:'a) = exists q (fun v -> v = x)
```

Complete your definition below, being sure that your type of `exists` is compatible with its uses above. We suggest that you implement this operation in Codio and then cut and paste your answer in the block below. (Codio answers will not be graded.)

```
let exists (q: 'a queue) (f : 'a -> bool) : bool =
  let rec loop qno =
    begin match qno with
      | None -> false
      | Some qn -> f qn.v || loop qn.next
    end
  in
  loop q.head
```

3. **Objects in OCaml**

Consider the following OCaml record type definition that is analogous to the interface `Iterator<E>` from the Java collections framework.

```
1  type 'a iterator = {
2     next     : unit -> 'a ;
3     has_next : unit -> bool
4  }
```

The function `q_iterator` below produces a value of this type, an iterator for the `queue` type from HW4. (Note, this function relies on OCaml's `ref` type, described in Appendix B).

```
5  let q_iterator (q : 'a queue) : 'a iterator =
6    let curr = { contents = q.head } in
7    {
8      has_next =
9        (fun () ->
10         begin match curr.contents with
11           | None   -> false
12           | Some _ -> true
13         end);
14     next =
15       (fun () ->
16        begin match curr.contents with
17          | None   -> failwith "empty queue"
18          | Some y -> (curr.contents <- y.next;
19                       y.v)
20        end)
21   }
```

(a) (2 points) The closest analogue in Java to the definition for `q_iterator` is

- ☐ an interface
- ☐ an abstract class
- ☒ a concrete class
- ☐ a set of fields

(b) (2 points) What is the OCaml type of the local variable `curr`, defined on line 6?

- ☐ `'a queue`
- ☐ `'a qnode ref`
- ☐ `'a qnode option`
- ☒ `'a qnode option ref`

(c) (2 points) The closest analogue in Java to the local variable `curr` would be a field declared as (assuming the appropriate definition of class `QNode<E>`):

- ☐ **`public static`** `QNode<E> curr;`
- ☐ **`private static`** `QNode<E> curr;`
- ☐ **`public`** `QNode<E> curr;`
- ☒ **`private`** `QNode<E> curr;`

(d) (2 points) Consider the following OCaml function that uses the iterator definition above.

```
let sum1 (it : int iterator) : int =
  let rec loop (acc: int) : int =
    if it.has_next () then
      loop (acc + it.next())
    else
      acc
  in
  loop 0
```

Is the `loop` function inside `sum1` *tail recursive*?

☒ Yes ☐ No

(e) (2 points) Consider the following OCaml function that uses the iterator definition above.

```
let sum2 (it : int iterator) : int =
  let rec loop () : int =
    if it.has_next () then
      let v1 = it.next() in
      let v2 = loop ()   in
      v1 + v2
    else
      0
  in
  loop ()
```

Is the `loop` function inside `sum2` *tail recursive*?

☐ Yes ☒ No

(f) (6 points) Now consider an extension of the `iterator` record type with a new operation, called `restart`. After this operation has been invoked, the next use of the `it.next()` should start producing the first value in the sequence.

What should be the type of this new operation? (Hint: in Java, the analogous modification would be to add the declaration `void restart();` to the `Iterator` interface.)

```
unit -> unit
```

What should be the definition of this operation in `q_iterator`? Fill in the implementation of the `restart` component, which should be added to the record between lines 20 and 21.

```
restart = fun () -> curr.contents <- q.head
```

Grading scheme:

- 2 - correct type above
- 2 - `fun () ->`
- 2 - `curr.contents <- q.head`

4. **OCaml and Java ASM concepts** (10 points)

Indicate whether the following statements are true or false by **typing an 'x' in the appropriate box**.

a. True ☒ False ☐

In OCaml, only record components declared as `mutable` may be updated in the heap at runtime.

b. True ☒ False ☐

In the OCaml ASM, stack bindings are immutable by default whereas in the Java ASM they are mutable by default.

c. True ☐ False ☒

In OCaml, if `s` and `t` are variables of type `int option`, and `s == t` returns `false`, then `s = t` will also return `false`.

d. True ☐ False ☒

In OCaml, all infinite loops will trigger a `Stack_overflow` runtime error.

e. True ☒ False ☐

In OCaml, a first-class function stored in the heap sometimes includes references to values of variables that were on the stack when the function was defined.

f. True ☐ False ☒

In Java, the `this` reference may be null.

g. True ☒ False ☐

In the Java ASM, the `this` reference is added to the stack when a nonstatic method is called.

h. True ☐ False ☒

In Java, it is impossible to create an alias to the `this` reference.

i. True ☐ False ☒

In Java, if `s` and `t` are variables of type `String` and `s == t` returns `false`, then `s.equals(t)` will also return `false`.

j. True ☐ False ☒

In the Java ASM, objects are stored on the stack.

5. **Array Design Problem**

In this problem, you will use the design process to implement a static method called `raggedBlend` in Java. You will need to read through Steps 1 and 2 carefully and answer questions to complete Steps 3 and 4.

Step 1: Understand the problem The `blend` operation, which you implemented in the Pennstagram assignment, combines two pictures by taking a weighted average of each pixel. In your homework, you only needed to consider the situation when the blended images were rectangular and had the same size.

For this problem, you will generalize that operation so that it works for a pair of "ragged" (i.e. non rectangular) 2-d arrays, which may not have the same size and shape. The output of this operation should also be a ragged 2-d array, with values only at coordinates where both input arrays have data.

Step 2: Design the interface For simplicity in this exam, we will work with 2-d arrays of `int`s instead of 2-d arrays of `Pixel`s. Therefore, your goal for this problem is to implement a static method with the following declaration.

```
public static int[][] raggedBlend(double alpha, int[][] a1, int[][] a2)
```

The interface of this method includes describing its behavior when given *invalid* inputs. First, the parameter `alpha` should be a double in the range from 0.0 to 1.0 (inclusive), indicating the weighting between the two arrays. If this parameter is outside of this range, then the method should throw an `IllegalArgumentException`.

Second, the two arguments `a1` and `a2` should be ragged 2-d arrays, *i.e.* each should be an array of arrays of ints. If `a1`, `a2`, or any reference in the arrays is null, then `raggedBlend` should throw an `IllegalArgumentException`.

IMPORTANT: Your implementation of `raggedBlend` **should *never* throw a `NullPointerException` no matter what arguments are provided**. Instead, it must detect that situation *before* it would occur and throw an `IllegalArgumentException` instead.

Step 3: Write test cases (9 points)

The next step is to write tests. For example, we can test that the method throws the right exception when the `alpha` parameter is out of range, with the following.

```
@Test void testInvalidAlpha() {
    Assertions.assertThrows(IllegalArgumentException.class, () -> {
        RaggedBlend.raggedBlend(-23, new int[0][0], new int[0][0]);
    });
}
```

Furthermore, we can test the output when given two arrays that contain no elements using this test.

```
@Test void testEmptyArrays() {
    int[][] empty = {{}};  // a 2-d array containing no elements
    assertArrayEquals(empty, RaggedBlend.raggedBlend(0,empty,empty));
}
```

Below, write more tests for `raggedBlend`. We want to see at least **three good, nonoverlapping tests** in addition to the examples above, but you may include more. You will be graded on correctness and comprehensiveness. If your test cases are too similar to eachother, you will lose points. We suggest that you first implement these tests in Codio and then cut and paste them into the block below.

The sorts of test cases that we are looking for include:

(a) If a1 is null or a2 is null, throws `IllegalArgumentException`

(b) If alpha is out of range, throws `IllegalArgumentException` (given test case).

(c) If there is some i where a1[i] is null, or a2[i] is null, throws `IllegalArgumentException`

(d) Two empty arrays: If a1 and a2 are both empty 2d arrays, result is empty 2d array (given test case).

(e) One empty array and one nonempty array: If a1 is nonempty array, a2 is empty array. result is empty 2d array.

(f) Two nonempty, rectangular arrays of the same size: If a1 contains a single element, and a2 contains a single element. result is array with a single element

(g) Two ragged arrays of the same shape.

(h) Two ragged arrays with different shapes. If a1 has two columns each with 1 element, a2 has two rows each with 1 element. result is array with 1 element.

(i) Two ragged arrays, of different shapes, where output is also ragged.

Three points per test up to nine points. Tests that duplicate the provided test cases above don't receive credit. Two points deducted for incorrect test case.

(If you need more space, you may use the scratch space at the end of the exam. But tell us if you do.)

Step 4: Implementation (16 points)

Now complete the implementation of `raggedBlend`. In your solution, you should use the following method to compute the weighted average of two `int`s.

```
public static int weightedAverage(double alpha, int x, int y) {
   return (int) Math.round(x * alpha + y * (1 - alpha));
}
```

You may also use the static method `Math.min` in addition to the method above. ***IMPORTANT****: You may not use any other methods from the Java libraries. Recall that accessing the length of an array is not a method call.* We suggest that you first implement this operation in Codio and then cut and paste your answer in the block below. (Codio answers will not be graded.)

```
public static int[][] raggedBlend(double alpha, int[][] a1, int[][] a2) {
```

```
1         if (a1 == null || a2 == null || alpha < 0 || alpha > 1.0) {
2             throw new IllegalArgumentException();
3         }
4         // check inner arrays nonnull
5         for (int i=0; i < a1.length; i++) {
6           if (a1[i] == null) {
7             throw new IllegalArgumentException();
8           }
9         }
10        for (int i=0; i < a2.length; i++) {
11          if (a2[i] == null) {
12            throw new IllegalArgumentException();
13          }
14        }
15
16        int min1 = Math.min(a1.length, a2.length);
17        int[][] out = new int[min1][];
18        for (int i=0; i< min1; i++) {
19            int min2 = Math.min(a1[i].length, a2[i].length);
20            out[i] = new int[min2];
21            for (int j=0; j<min2; j++) {
22                out[i][j] = weightedAverage(alpha,a1[i][j], a2[i][j]);
23            }
24        }
25        return out;
26    }
```

Instead of lines 8-17 above, we also accepted solutions that only check the arrays at positions that are actually accessed by the method. In other words, replacing these lines with the following inside the first-level for loop.

```
if (a1[i] == null || a2[i] == null) {
    throw new IllegalArgumentException();
}
```

(If you need more space, you may use the scratch space at the end of the exam. But tell us if you do.)

6. **Java Typing, Inheritance, and Dynamic Dispatch** (16 points total)

Consider the Java classes shown in Appendix D and Appendix E (or in the Java Codio project in the `Game` folder). These classes are inspired by a *very* simplified version of the game *Mushroom of Doom*.

(a) (3 points) Consider the following local variable declaration (which does not appear in the provided code):

```
_________________ snitch = new Circle(1,3,10);
```

In the box below, list **all** types (**there may be one or more**) that can be used for the declaration of `snitch` above. You may assume that `snitch` is not used anywhere else in the program.

`Circle,GameObject,Object`

(b) (3 points) Consider the following local variable declaration (which does not appear in the provided code):

```
List<List<Circle>> allGames = new LinkedList<List<Circle>>();
_________________ iter = allGames.iterator();
```

In the box below, list **all** types (**there may be one or more**) that can be used for the declaration of `iter` above. You may assume that `iter` is not used anywhere else in the program. (For reference, documentation for the `List` and `Iterator` interfaces appears in Appendix C.)

`Iterator<List<Circle>>,Object`

(c) (5 points) Which method call(s) below must use dynamic dispatch to resolve the appropriate code to place on the workspace of the ASM? (Select **all** appropriate answers.)

- ☐ `super.paintComponent(g);` (Line 20 of `GameCourt`)
- ☒ `go.draw(g);` (Line 22 of `GameCourt`)
- ☐ `Circle.random();` (Line 32 of `GameCourt`)
- ☒ `gameObjects.add(circle);` (Line 33 of `GameCourt`)
- ☐ `SwingUtilities.invokeLater(new Game());` (Line 46 of `Game`)

(d) (4 points) Which of the objects depicted below is allocated in the heap when the following code is placed on the workspace of the Java ASM:

```
GameObject circle = new Circle(1,2,9);
```

Select **one** option from the choices below, ignoring any members from the `Object` class.

☒ 4

Circle	
x	1
y	2
radius	9

☐ 2

Circle	
x	1
y	2

☐ 3

Circle	
radius	9

☐ 1

Circle

☐ 3

GameObject	
x	1
y	2
radius	9

☐ 2

GameObject	
x	1
y	2

Grading scheme:

- 1 point for correct class (Circle)
- 1 point for correct x, y (present)
- 2 points for correct radius (present)

(e) (4 points) Which of the objects depicted below is allocated in the heap when the following code is placed on the workspace of the Java ASM.

```
GameObject square = new Square(0,0);
```

Select **one** option from the choices below, ignoring any members from the `Object` class.

☐ 2

Square	
x	0
y	0
HEIGHT	10

☒ 4

Square	
x	0
y	0

☐ 1

Square	
HEIGHT	10

☐ 3

Square

☐ 1

GameObject	
x	0
y	0
HEIGHT	10

☐ 3

GameObject	
x	0
y	0

Grading scheme:

- 1 point for correct class (Square)
- 1 point for correct x, y (present)
- 2 points for correct HEIGHT (not present)

7. **Swing and Event Handlers**

These questions concern the `GameCourt` and `Game` classes shown in Appendix E (and available in the Java Codio project in the `Game` folder).

(a) (2 points) What is `ActionListener` referred to on line 22 of the `Game` class? (Select one answer.)

- ☐ a class defined in Swing
- ☒ an interface defined in Swing
- ☐ a class defined in Appendix E
- ☐ an interface defined in Appendix E
- ☐ a method for the class `JButton`
- ☐ a constructor for the class `JButton`

(b) (2 points) What is `MouseAdaptor` referred to on line 29 of the `Game` class? (Select one answer.)

- ☒ a class defined in Swing
- ☐ an interface defined in Swing
- ☐ a class defined in Appendix E
- ☐ a concrete class defined in Appendix E
- ☐ an interface defined in Appendix E
- ☐ a method for the class `GameCourt`
- ☐ a constructor for the class `GameCourt`

(c) (2 points) There are eight occurrences of the **`new`** keyword in the `run` method of the class `Game`. How many of them correspond to anonymous inner classes? (Your answer should be in the range 0 - 8.)

2

(d) (3 points) In a short sentence, describe the effect of the DoIt! button from the user's point of view.

ANSWER: A new circle appears in the `GameCourt` *panel, in a random location. Partial credit: runs the court.doIt function when clicked.*

(e) (4 points) The `getPoints` method in the `GameObject` class determines whether an object has been hit, *i.e.* whether the provided coordinates are within the bounds of the object. This method returns 10 points for a hit and 0 points otherwise.

In the box below, add new code to the `Square` class so that hits are worth -10 points instead of 10 points. Your code should not duplicate any of the functionality of the `GameObject` class.

```
@Override
public int getPoints(int x0, int y0) {
     return -1 * super.getPoints(x0,y0);
}
```

Grading notes:

- Must override `getPoints` method, otherwise the points for hitting a `Square` will not change.
- Must not duplicate calculation of `inBounds`, should call method from superclass instead. (We are also looking for a correct call to the superclass method.)
- No deduction if override tag is missing

(f) (12 points) The provided code is not much of a game as there no way to score points!

To make this game (slightly) more fun, rewrite the `click` method in the `GameCourt` class so that it does the following.

- It should determine which of the game objects were hit (if any) and calculate the total points from those hits. (Note, nothing prevents game objects from overlapping, so a single click could hit multiple objects.) You should assume that this modification is in addition to the previous part, so Squares should be worth -10 points when hit.
- If no game objects were hit, then this method should add a new `Square` to the playing field, in a random location.

After your update, when the player uses the mouse to click on a shape, they should immediately be able to see the updated score at the bottom of the window. Or, if they click anywhere else in the playing field, they should immediately see the new square.

```
public int click(int x0, int y0) {

    boolean anyHit = false;
    int points = 0;
    for (GameObject obj : gameObjects) {
        int objPoints = obj.getPoints(x0,y0);
        if (objPoints != 0) {
           anyHit = true;
        }
        points = points + objPoints;
    }
    if (!anyHit) {
        gameObjects.add(Square.random());
        this.repaint();
    }
    return points;
}
```

Grading notes:

- The `gameObjects` list is implemented with a linked list, so the correct way to iterate over this structure is using an iterator. Use of index based access, such as with `gameObjects.get` is not appropriate for this data structure.
- The game state only changes when the square is added, so that is the only place where repaint needs to be called. (No deduction if it is always called.)

(End of main exam.)

Scratch page. Add any additional answers below, but be sure to tell us to look here!

CIS 120 Final Exam — Appendices

A OCaml queue implementation

```
type 'a qnode = { v: 'a;
                  mutable next: 'a qnode option }

type 'a queue = { mutable head: 'a qnode option;
                  mutable tail: 'a qnode option }

(* INVARIANT:
 - q.head and q.tail are either both None, or
 - q.head and q.tail both point to Some nodes, and
   - q.tail is reachable by following 'next' pointers
     from q.head
   - q.tail's next pointer is None
*)

let create () : 'a queue =
  { head = None; tail = None }

(* Add an element to the tail of a queue *)
let enq (elt: 'a) (q: 'a queue) : unit =
  let newnode = { v = elt; next = None } in
  begin match q.tail with
  | None ->
     (* Note that the invariant tells us that q.head is also None *)
     q.head <- Some newnode;
     q.tail <- Some newnode
  | Some n ->
     n.next <- Some newnode;
     q.tail <- Some newnode
  end

(* Remove an element from the head of the queue *)
let deq (q: 'a queue) : 'a =
  begin match q.head with
  | None ->
     failwith "deq called on empty queue"
  | Some n ->
     q.head <- n.next;
     if n.next = None then q.tail <- None;
     n.v
  end
```

B OCaml Ref Type

```
type 'a ref = {mutable contents : 'a}
```

We can construct a value of type `int ref` with the notation `let x = contents = 3`. After this definition, the notation `x.contents` access the int stored in the reference and the notation, `x.contents <- 5` updates the int stored in the reference to be the value 5.

C Java documentation

Below, we summarize some of the interfaces found in the exam. You may also refer to the online Java 8 documentation for the standard library (`https://docs.oracle.com/javase/8/docs/api/java/lang/package-summary.html`), the collections library (`https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html`) and for Swing (`https://docs.oracle.com/javase/8/docs/api/javax/swing/package-summary.html`).

C.1 Java `Iterator<E>` interface

```
interface Iterator<E>

boolean hasNext()
         // Returns true if the iteration has more elements.

E       next()
         // Returns the next element in the iteration.
```

C.2 Java `List<E>` interface

```
interface List<E>

boolean add(E e)
         // Appends the specified element to the end of this list.

void    clear()
         //Removes all of the elements from this list.

boolean contains(Object o)
         //Returns true if this list contains the specified element.

Iterator<E>     iterator()
         //Returns an iterator over the elements in this list in proper sequence.
```

C.3 Java `Runnable` interface

```
interface Runnable

void run()
     // When an object implementing interface Runnable is used to create a
     // thread, starting the thread causes the object's run method to be
     // called in that separately executing thread.
```

D Java Code: Game Objects

GameObject.java

```
4  public abstract class GameObject {
5      private int x; // upper-left corner of object
6      private int y;
7
8      public GameObject(int x0, int y0) {
9          x = x0;
10         y = y0;
11     }
12
13     public int getX() {
14         return x;
15     }
16
17     public int getY() {
18         return y;
19     }
20
21     abstract void draw(Graphics g);
22
23     abstract public int getWidth();   // bounds of the object
24     abstract public int getHeight();
25
26     public int getPoints(int x0, int y0) {
27         boolean inBounds = (x <= x0 && x0 <= x + getWidth()
28                             && y <= y0 && y0 <= y + getHeight());
29         if (inBounds) {
30             return 10;
31         } else {
32             return 0;
33         }
34     }
35
36 }
```

Circle.java

```
5  public class Circle extends GameObject {
6      private int radius;
7
8      public Circle(int x0, int y0, int r0) {
9          super(x0,y0);
10         this.radius = r0;
11     }
12
13     @Override
14     public void draw(Graphics g) {
15         g.fillOval(this.getX(), this.getY(), this.radius, this.radius);
16     }
17
18     public static Circle random() {
19         int r = 5 + new Random().nextInt(10);
20         int x = new Random().nextInt(GameCourt.COURT_WIDTH - r);
```

```
21          int y = new Random().nextInt(GameCourt.COURT_HEIGHT - r);
22          return new Circle(x,y,r);
23      }
24
25      @Override
26      public int getWidth() {
27         return radius;
28      }
29
30      @Override
31      public int getHeight() {
32          return radius;
33      }
34  }
```

Square.java

```
 5  public class Square extends GameObject {
 6      public static final int HEIGHT = 10;
 7
 8      public Square(int x0, int y0) {
 9          super(x0, y0);
10      }
11
12      @Override
13      public void draw(Graphics g) {
14          g.fillRect(this.getX(), this.getY(), HEIGHT, HEIGHT);
15      }
16
17      public static Square random() {
18          int x = new Random().nextInt(GameCourt.COURT_WIDTH - HEIGHT);
19          int y = new Random().nextInt(GameCourt.COURT_HEIGHT - HEIGHT);
20          return new Square(x,y);
21      }
22
23      @Override
24      public int getWidth() {
25          return HEIGHT;
26      }
27
28      @Override
29      public int getHeight() {
30          return HEIGHT;
31      }
32  }
```

E Java Code: Game GUI

GameCourt.java

```
 9 public class GameCourt extends JComponent {
10     public static final int COURT_WIDTH = 300;
11     public static final int COURT_HEIGHT = 300;
12
13     private List<GameObject> gameObjects = new LinkedList<GameObject>();
14
15     public GameCourt() {
16     }
17
18     @Override
19     public void paintComponent(Graphics g) {
20         super.paintComponent(g);
21         for (GameObject go : gameObjects) {
22             go.draw(g);
23         }
24     }
25
26     @Override
27     public Dimension getPreferredSize() {
28         return new Dimension(COURT_WIDTH,COURT_HEIGHT);
29     }
30
31     public void doit() {
32         GameObject circle = Circle.random();
33         gameObjects.add(circle);
34         this.repaint();
35     }
36
37     public int click(int x0, int y0) {
38         return 0;
39     }
40
41 }
```

Score.java

```
 6 public class Score extends JPanel {
 7     private int score = 0;
 8     private JLabel scoreText = new JLabel();
 9
10     public Score() {
11         this.add(new JLabel("Score:"));
12         this.add(scoreText);
13         scoreText.setText(Integer.toString(score));
14     }
15
16     public void add(int points) {
17         score = score + points;
18         scoreText.setText(Integer.toString(score));
19         scoreText.repaint();
20     }
21 }
```

Game.java

```
6  public class Game implements Runnable {
7
8      public void run() {
9          JFrame frame = new JFrame("Top-level Frame");
10         JPanel panel = new JPanel();
11         panel.setLayout(new BorderLayout());
12         frame.getContentPane().add(panel);
13
14         final GameCourt court = new GameCourt();
15         final Score score = new Score();
16         final JButton button = new JButton("Do It!");
17
18         panel.add(button, BorderLayout.PAGE_START);
19         panel.add(court,  BorderLayout.CENTER);
20         panel.add(score,  BorderLayout.PAGE_END);
21
22         button.addActionListener(new ActionListener() {
23             @Override
24             public void actionPerformed(ActionEvent e) {
25                 court.doit();
26             }
27         });
28
29         court.addMouseListener(new MouseAdapter() {
30             @Override
31             public void mouseClicked(MouseEvent e) {
32                 // access location of mouse click
33                 int x = e.getX();
34                 int y = e.getY();
35                 int points = court.click(x, y);
36                 score.add(points);
37             }
38         });
39
40         frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
41         frame.pack();
42         frame.setVisible(true);
43     }
44
45     public static void main(String[] args) {
46         SwingUtilities.invokeLater(new Game());
47     }
48 }
```