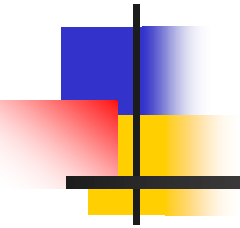
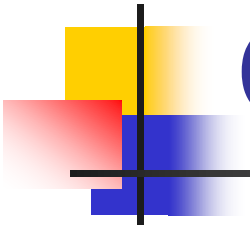
ANALYZING ASPECTS of a GAMING APPLICATION



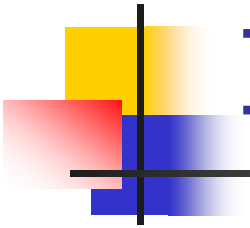
by

Cem AKSOY
Mücahid KUTLU
Bahri TÜREL



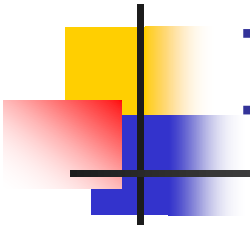
Outline

- Introduction
- Problem Statement
- Requirements Analysis
- Identified Aspects
- Aspect-Oriented Program & Demo
- Specification with JBoss
- Conclusion
- Question&Answers



Introduction

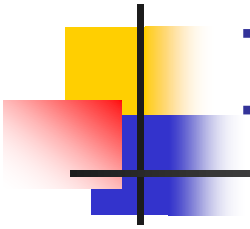
- Wide range of usage area of Gaming Applications
- One of the well improved area of computer application
- Improvements going on
- E.g. Object-Oriented programming, modularization with OpenGL



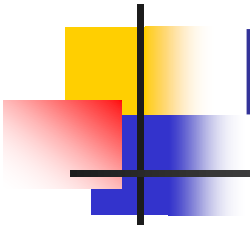
Introduction(Cont'd)

- Seperation of concerns aimed by modularization
- High Coherence & Loose Coupling
- Increasing complexity disables seperation of concerns
- OO lacks to modularize the system fully
- Collision detection is a good example of the complexity

Introduction(Cont'd)



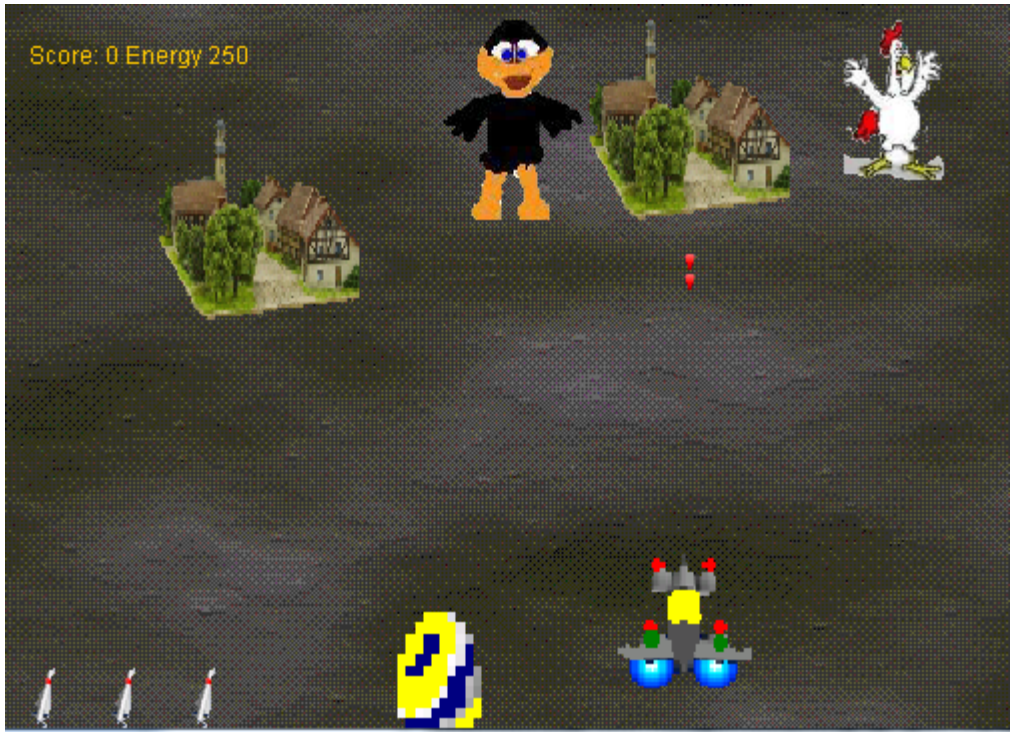
- Alternative approach is AOP for gamig applications
- Identification of key concerns leads aspects
- Identifying good aspects prevents crosscutting concerns
- Easier modularization and development is possible with AOP

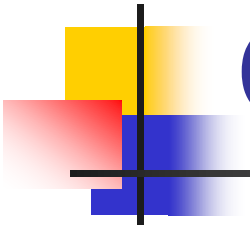


Problem Statement

- Apply AOP on a pre-developed OO arcade game, Chicken Invaders
- Single player, level based game
- Player tries to finish the levels by using a spaceship
- Kill enemies and try not to die
- Considerable amount of different types of objects
- Considerable amount of collisions and interactions between these objects

Problem Statement(Cont'd)

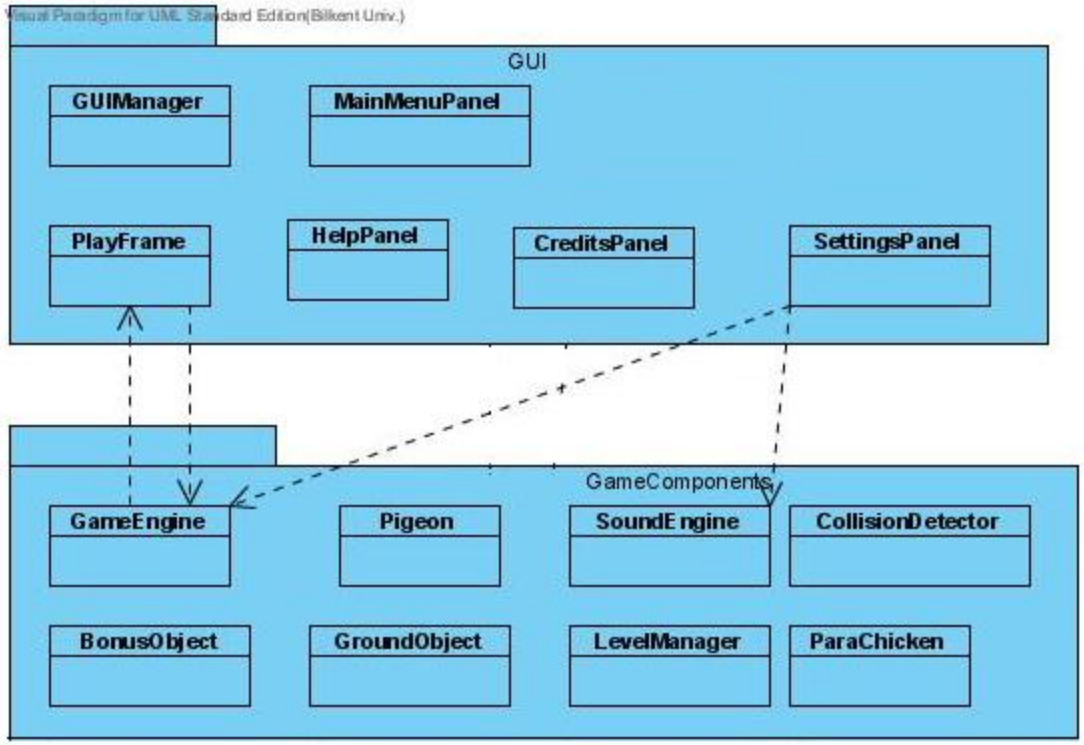


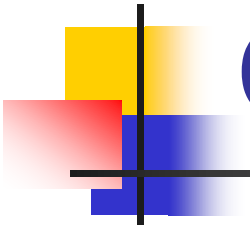


OO Design of Game

- Two packages available: *GUI* and *GameComponents*
- *GUI* responsible for user interface and graphics
- *GameComponents* responsible for game objects

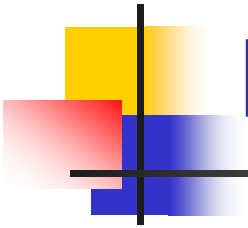
OO Design of Game(Cont'd)





OO Design of Game(Cont'd)

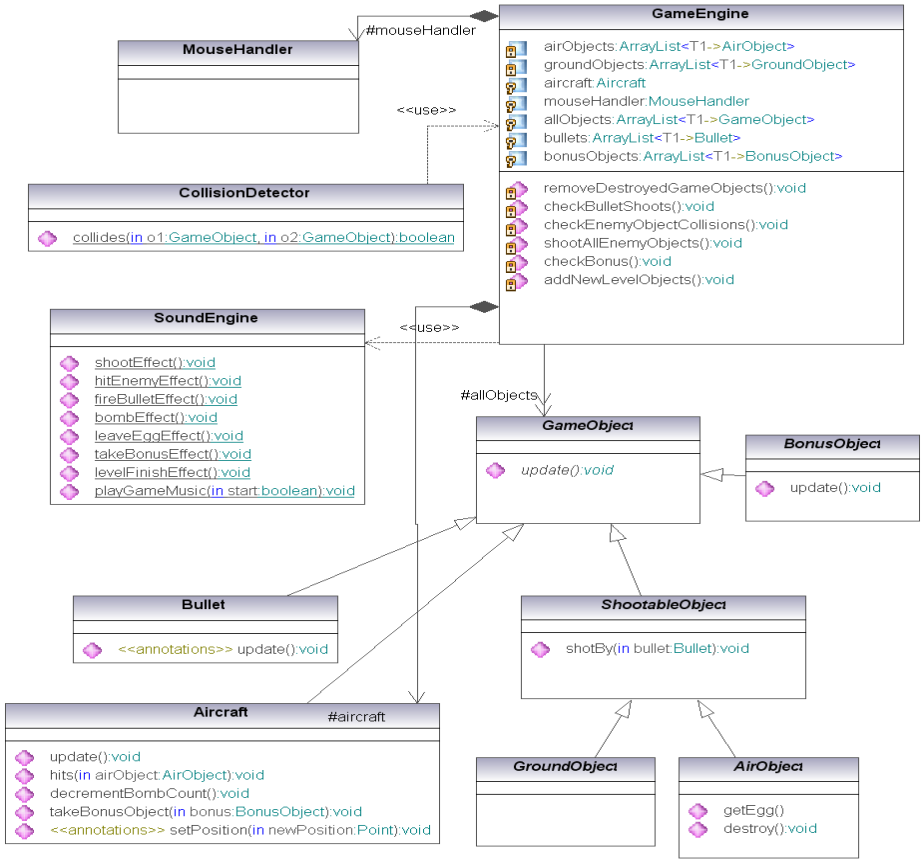
- Dependencies between few classes of different packages
- Main interactions between *PlayFrame* and *GameEngine*
- *GameEngine* triggers *PlayFrame* to draw all the objecs in each interval
- *SettingsPanel* changes music level and game speed

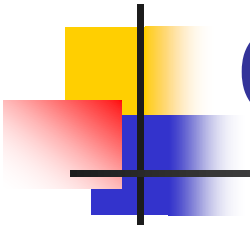


Responsibilities of *GameEngine*

- *GameEngine* main controller class
  - Controls flow of the game
  - Collision detection
  - Object creation,update and destruction
  - Level manaegment
  - Running *SoundEngine* in proper times
- Too many different types of responsibilities!

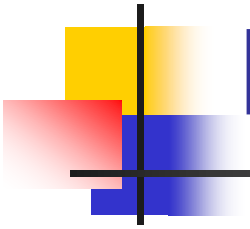
GameComponents Package





Classes of *GameComponents*

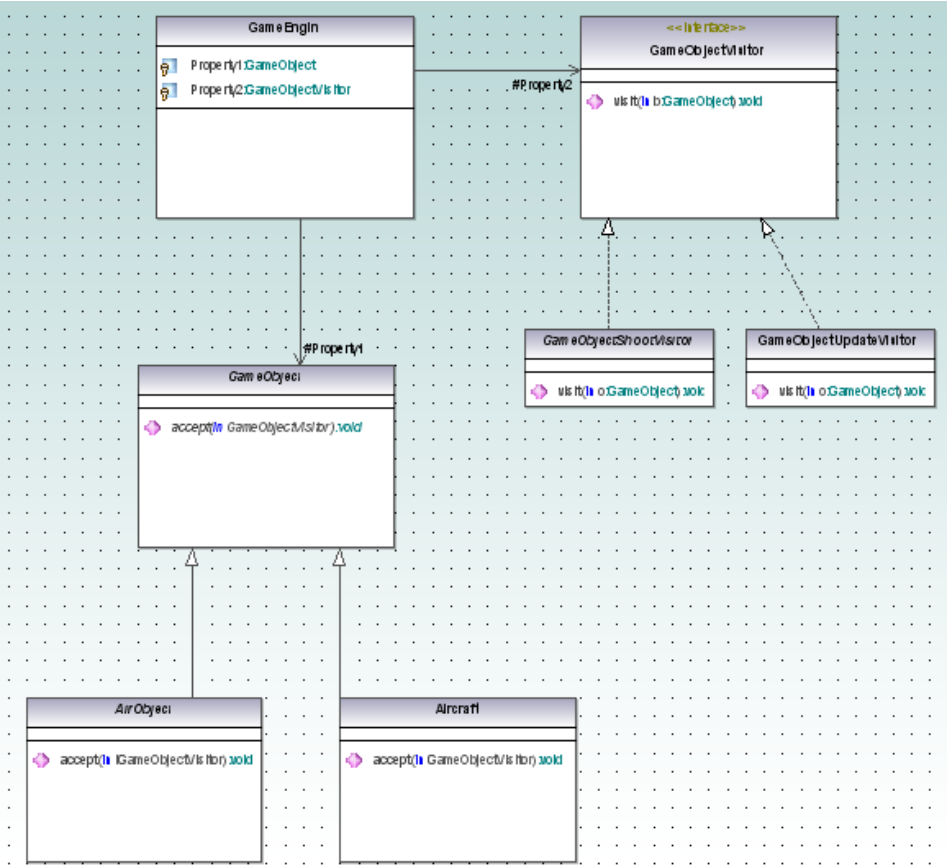
- Two types of general classes: *BonusObject*, *ShootableObject*
- *BonusObject* cannot be shot, disappear when collision with aircraft occurs
- Two Types of S*hootableObject* *GroundObject* and *AirObject*
- All other game objects like *AirCraft*, *Bullet* extends *GameObject*



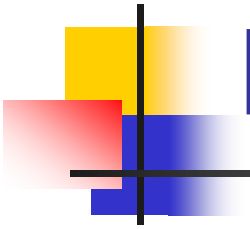
Design Patterns

- Singleton
  - GameEngine class
- Visitor
  - GameObjectUpdateVisitor
  - GameObjectShootVisitor

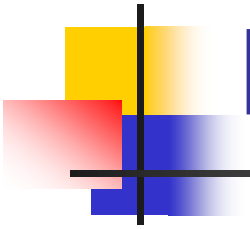
Design Patterns (cont'd)



General Problems with OO Design



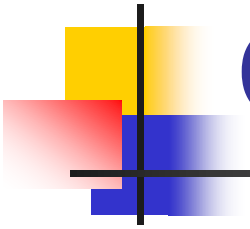
- Some concerns cannot be modularized well and they crosscut over multiple modules
  - Like sound events
- Some objects may become non-coherent due to big responsibility load
  - Like *GameEngine*



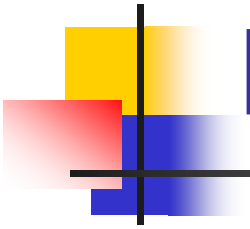
Requirements Analysis

- Well-known requirements of every game
- Player plays the game
- Player can change settings of the game

Identifying Crosscutting Concerns



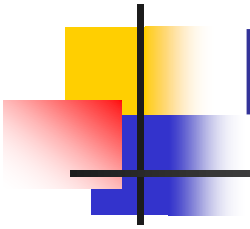
- Main Concerns needed to be identified:
  - Detecting Collisions
  - Removing out of frame objects
  - Playing Sound
  - Throwing Bomb



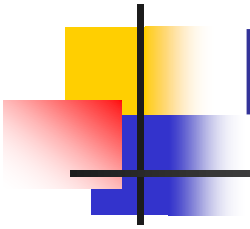
Detecting Collisions

- Collision between *AirCraft* and *AirObject* and between *Bullet* and *ShootableObject* occurs
- Detecting these collisions is significant for the flow of the game

Detecting Collusions(Cont'd)



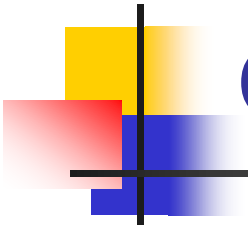
- GameEngine checks these collisions
- *checkBulletShoots()* to detect collision of bullets and *ShootableObjects*
- *checkBonus()* to detect collision of aircraft and bonus objects
- checkEnemyObjectCollisions() method to detect collision of aircraft and air objects



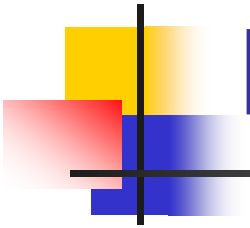
Detecting Collisions(Cont'd)

- These three methods use *CollisionDetector* class's *collides(GameObject,GameObject)* method
- Handling collision detection concern in *GameEngine* class causes low coherency

Removing out of frame objects



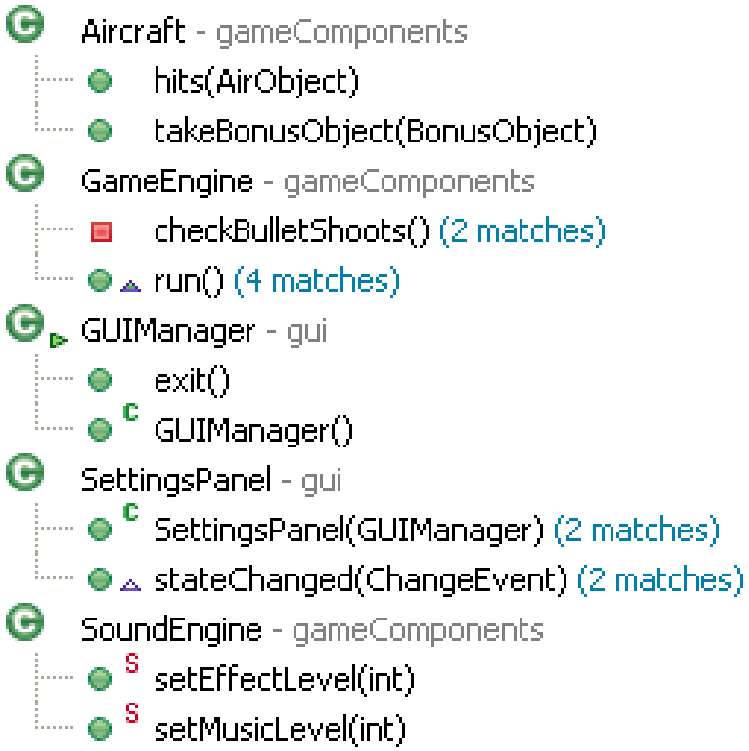
- Enemy objects, bonus packages move from up to down
- Bullets move from down to up
- Out of frame objects need to be removed from object list
- Having updated location after each move check if out of frame
- RemoveDestroyedGameObjects() does this control

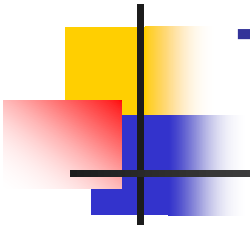


Playing Sound

- Different sound effects are used for some specific actions
  - a crash, shooting an enemy object with a bullet or using a bomb
- *SoundEngine* coordinates all sound effects by using different methods for each type of sound
- These methods are called by different classes (scattering concern and maintenance problem)

Playing Sound(Cont'd)

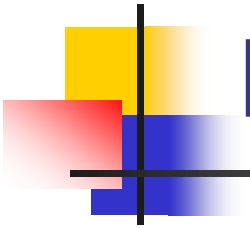




Throwing Bomb

- Bomb decreases health of all game objects that are currently in frame by a certain amount
- Handled in *GameEngine* with shootAllEnemyObjects() and in *Aircraft* with decrementBombCount()
- Also handled in *GameEngine* class

Aspect-Oriented Solution of Detecting Collision Concern

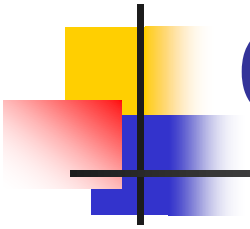


- Collusion between AirCraft and Air/BonusObject
  - Pointcut selects joinpoints where position of aircraft is changed
  - For each air object that is hold in GameEngine, we perform collision check
  - If there is a collision, health of air craft decreased and other object removed

Collision between AirCraft and Air/BonusObject

```
1  pointcut updateInPositions( Aircraft a, GameEngine g):
2          target(a) && this(g) && call( void Aircraft.setPosition(..));
3  after(Aircraft a,GameEngine g):updateInPositions(a, g){
4      for (AirObject ao : g.airObjects){
5          if (CollisionDetector.collides(a, ao)){
6              a.hits(ao);
7              ao.destroy();
8          }
9      }
10     for (int bi = 0; bi < g.bonusObjects.size(); bi++){
11         BonusObject bo = g.bonusObjects.get(bi);
12         if (CollisionDetector.collides(bo, a)){
13             a.takeBonusObject(bo);
14             g.bonusObjects.remove(bo);
15             g.allObjects.remove(bo);
16         }
17     }
18 }
```

Collision of Bullet and Game Objects

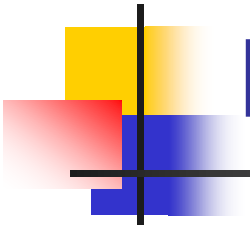


- Pointcut selects joinpoints where position of bullet is changed
- Check collision between bullet and shootable objects in advice part
- If collision is found, we call visit() method of GameObjectShootVisitor

Collision of Bullet and Game Objects

```
17 pointcut bulletCollision(Bullet b, GameEngine g):
18     args(b) && this(g) && call(void GameObjectUpdateVisitor.visit(GameObject));
19
20 after( Bullet b, GameEngine g ):bulletCollision(b, g){
21     for (int i = 0; i < g.airObjects.size(); i++) {
22             if (CollisionDetector.collides(g.airObjects.get(i), b)) {
23                 visitor.visit(g.airObjects.get(i));
24
25             }
26     }
27     for (int i = 0; i < g.groundObjects.size(); i++) {
28             if (CollisionDetector.collides(g.groundObjects.get(i), b)) {
29                 visitor.visit(g.groundObjects.get(i));
30             }
31     }
32 }
```

Aspect-Oriented Solution of Removing Out-of-Frame Objects

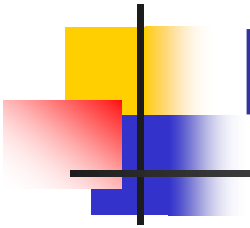


- Pointcut selects joinpoints where position of any GameObject is changed
- Get boundary of frame and check if out of frame, if so remove the object from proper lists in advice part

Aspect-Oriented Solution of Removing Out-of-Frame Objects

```
1  pointcut RemoveOutOfFrameObjects(GameEngine g,GameObject o) :
2      target(o) && this(g) && call ( void GameObjectUpdateVisitor.visit(..) );
3
4  after ( GameEngine g,GameObject o ) : RemoveOutOfFrameObjects(g,o){
5
6      double screenWidth = g.playFrame.getBounds().getWidth();
7      double screenHeight = g.playFrame.getBounds().getHeight();
8
9      Rectangle oRect = o.getRectangle();
10         if ((oRect.x + (oRect.width / 2)) < -10
11                 || (oRect.y + (oRect.height / 2)) < -10
12                 || ((oRect.y) - screenHeight) > 5
13                 || (oRect.x - screenWidth) > 5) {
14
15             g.allObjects.remove(o);
16             g.bonusObjects.remove(o);
17             g.airObjects.remove(o);
18             g.groundObjects.remove(o);
19         }
```

Aspect-Oriented Solution of Playing Sound

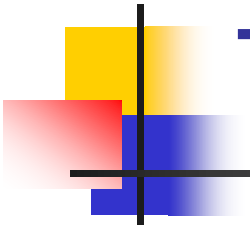


- There are several advices for playing sound aspect
- By having this aspect, we can easily manage playing sound concern like adding new sounds or cancelling a sound without searching in codes

Aspect-Oriented Solution of Playing Sound

```
7   pointcut soundAircraft() : call (void Aircraft.hits(..));
8
9   pointcut aspectGUIManager() :  call (public gui.GUIManager.new() );
10
11  after():soundAircraft()
12  {
13      SoundEngine.hitEnemyEffect();
14  }
15
16  after() : aspectGUIManager()
17  {
18      SoundEngine.loadSounds();
19  }
```

Aspect-Oriented Solution of Throwing Bomb



- Poincut selects joinpoints where number of bombs of aircraft is decremented by method decrementBomb()
- We call shotBy(Bomb) method of each object in game for advice

Aspect-Oriented Solution of Throwing Bomb

```
14    pointcut throwBomb(GameEngine e) :
15        this(e) && call ( void Aircraft.decrementBombCount() );
16
17    after(GameEngine e) : throwBomb(e)
18    {
19        for (int i = 0; i < e.airObjects.size(); i++) {
20            e.airObjects.get(i).shotBy(new Bomb());
21        }
22        for (int i = 0; i < e.groundObjects.size(); i++) {
23            e.groundObjects.get(i).shotBy(new Bomb());
24        }
25    }
```

Development Aspects

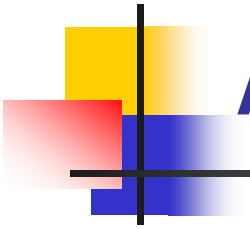
- Checking Creation of Game Objects
  - Pointcut selects constructor joinpoints of each class which is a sub-class of GameObject
  - Advice method prints message for creation of the new object by using thisjoinpoint

```
4  pointcut objectIsCreated(): call ( public GameObject+.new(..) );
5  after():objectIsCreated()
6  {
7      System.out.println(thisJoinPoint.toShortString()+" is created");
8  }
```

Development Aspects (Cont'd)

- Checking Deletion of Game Objects
  - Pointcut selects joinpoints in calling remove(GameObject) methods in GameEngine class and RemoveOutOfframe aspect
  - We give proper message about which object is deleted in advice part

```
1 pointcut objectIsRemoved(GameObject g) :
2         args(g) && call(* *.remove(..)) &&
3         (within(gameComponents.GameEngine) ||
4          within(gameComponents.GamaAspect));
5 after(GameObject g): objectIsRemoved(g)
6 {
7     System.out.println(g.getClass() + " is removed");
8 }
```



Alternative Approach: JBoss

- JBoss is a Java Aspect-oriented framework
- Everything in AspectJ can also be implemented in JBoss
- Playing sound concern is implemented in JBoss for a concrete example
- Syntax changes a little and logic is the same

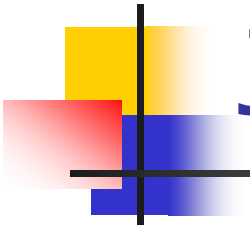
JBoss Example

a)

```
<bind pointcut="call(void gameComponents.Aircraft->;hits(..) )">
    <after aspect="src.gameComponents.JBossSound" name="AircraftCrash"/>
</bind>
```

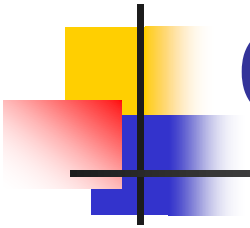
b)

```
public Object AircraftCrash( Invocation invocation  ) throws Throwable
{
    SoundEngine.hitEnemyEffect();
    return invocation.invokeNext();
}
```



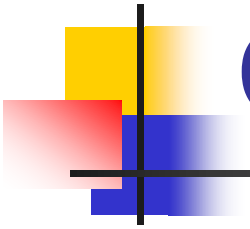
JBoss More

- Change program without recompilation by using JBoss
- Since pointcuts are defined in XML files
- In case of having a bug in a game, debug by changing advices or at least deactivate bugged module by deleting proper advice from XML



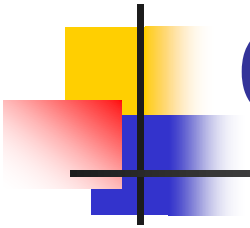
Conclusion

- Gaming applications are complex systems to develop
- OOP is a good discipline but lacks in crosscutting concerns
- Alternative approach is AOP
- Focused on aspects by searching concerns that crosscut



Conclusion(Cont'd)

- Complex concerns like detecting collision, removing out-of-frame objects, playing sound and throwing bomb are easily handled with the help of AOP
- Maintenance is much more simpler that OOP



Questions&Answers

- Thanks for your listening
- Any Questions?