CSSE 220

Collision Handling without **instanceof**

Checkout *InheritanceDesign* project from public Git

The specific problem

- Players can blow Bubbles.
- Bubbles can capture monsters.
- Players can pop ‘Bubbled’ monsters.
- Players can be killed by free monsters.
- Players can take powerups.

So many collisions! How do we handle them all?

Handling Collisions in General

- Many ways it can be done
- Good design principles
  - Make it **easy to use and extend code**
- Functional but bad design principles
  - Use “type predicated code”
  - … if (“instanceof, getClass(), getType()” ==/equals)
  - OUTLAWED in your project design!
  - Your design should not use these at all!

Live Coding

- Let’s consider a simulator with collisions
- More Raindrops More Platforms
- Run Main.java

Extend Functionality

- Implement an InvincibilityDrop
  - makes a BouncingPlatform invincible for 50 ticks
  - drops should not affect platforms when invincible
  - color should be Yellow
  - size should be 20
  - absorbed by UserControlledPlatform like others

GameComponent.java

- Let’s look how collisions are handled
- Where would changes in code need to happen?
- Look for examples of “type predicated code”
  - Why is this hard to use/extend?

Collision Chart

When do collisions happen?

	UserPlat	BouncePlat	DamageDrop	HealDrop
UserPlat				
BouncePlat				
DamageDrop				
HealDrop				

Collision Chart

When do collisions happen?

	UserPlat	BouncePlat	DamageDrop	HealDrop
UserPlat			X	X
BouncePlat			X	X
DamageDrop	X	X		
HealDrop	X	X		

Collision Chart

When we have multiple objects that interact with the same object, we can use ***inheritance*** to help us make a design to deal with the collisions. We should have the classes that differ provide the functionality to deal with the collisions.

	UserPlat	BouncePlat	DamageDrop	HealDrop
UserPlat			X	X
BouncePlat			X	X
DamageDrop	X	X		
HealDrop	X	X		

Collision Chart

When we have multiple objects that interact with the same object, we can use ***inheritance*** to help us make a design to deal with the collisions. We should have the classes that differ provide the functionality to deal with the collisions.

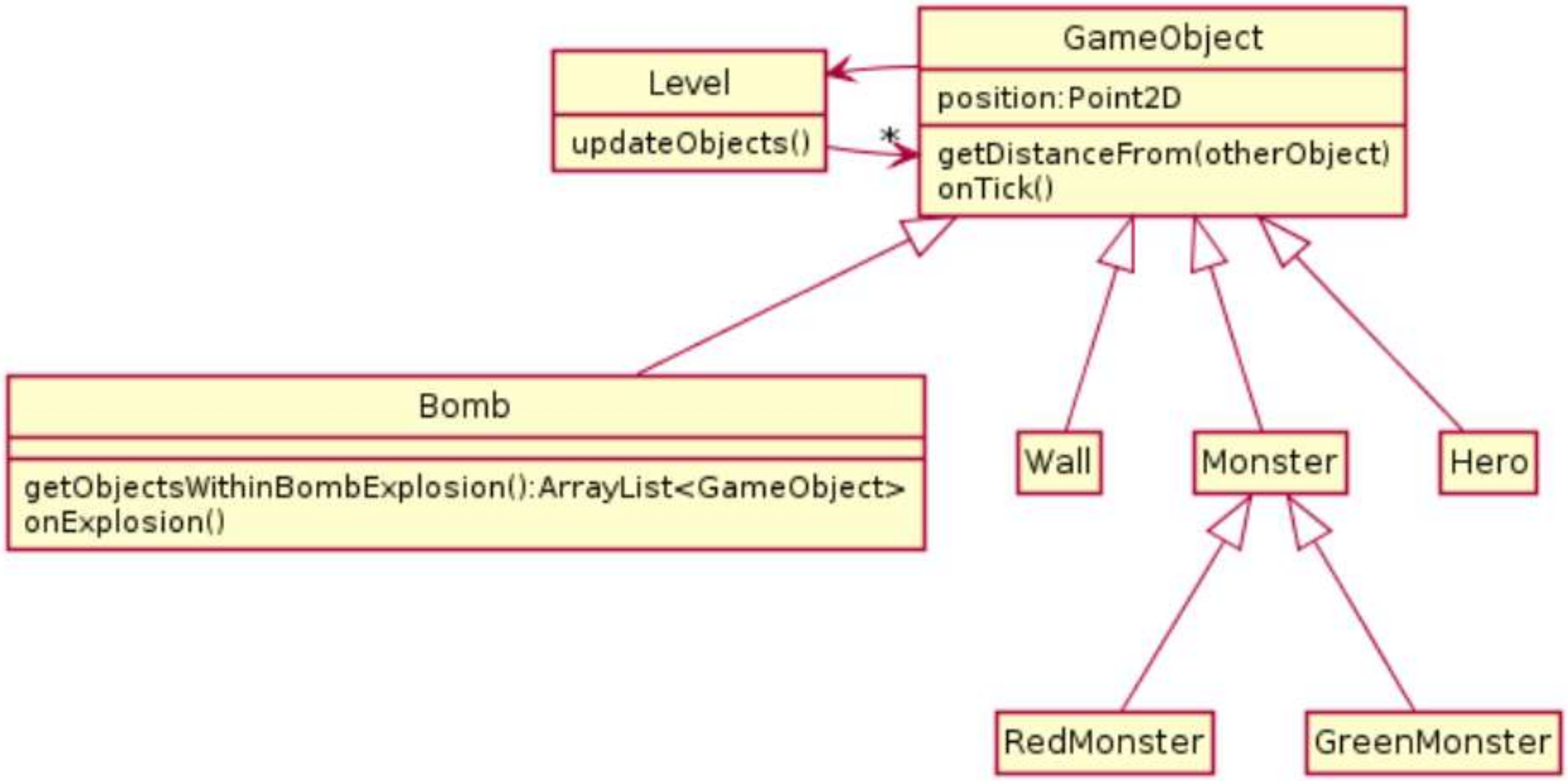
So in this case, the different drops (and InvincibilityDrop) differ in how they affect the platform. Thus, we can could create an abstract class and require an implementation of the collideWith(BouncingPlatform) method. Similarly, we can provide a collideWith(UserControlledPlatform) method, this will allow us to put the code in the drop classes and make the processing MUCH cleaner as we will see.

Live-coding

Design Activity

- Hand out

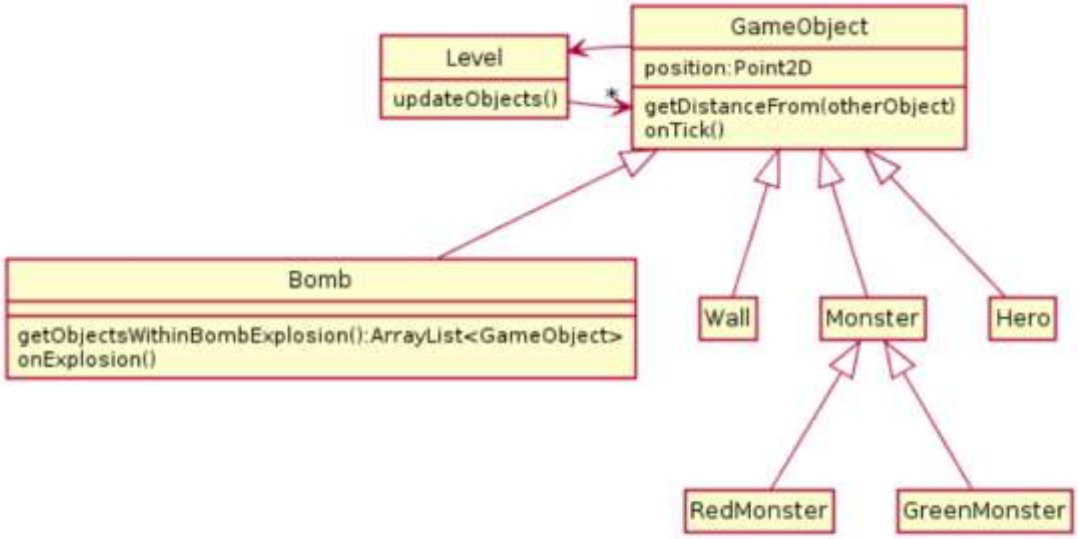
In the game Bomberman, everything has special behavior if caught in a bomb explosion. Heroes die and restart the level, monsters are killed and score points, walls are destroyed, and bombs explode themselves. In the design below, the Bomb class has an onExplosion method that handles this behavior and (you can assume) works correctly.



Collision Code

```
for(GameObject g : getObjectsWithinBombExplosion()) {

if(g instanceof Bomb) {
   // bomb specific code
}
if(g instanceof Wall) {
   //wall specific code
}
//pattern continues...
}
```



1. What is wrong with the design?
2. Propose a new design that does not have the problem you identified in #1. You only need include in your UML diagram classes that are *different* from their version in the given diagram.
Also include a sample for what the analogous code on onExplosion looks like

Work time

Be sure everyone is getting a chance to drive.

TEAM PROJECT

Platforms and Drops

- What if we wanted Platforms to collide as well?

	UserPlat	BouncePlat	DamageDrop	HealDrop
UserPlat		X	X	X
BouncePlat	X	X	X	X
DamageDrop	X	X		
HealDrop	X	X		

Platforms and Drops

- AbstractPlatform and AbstractDrop
  - Subclasses implement
  - collideWith(AbstractPlatform)

	UserPlat	BouncePlat	DamageDrop	HealDrop
UserPlat		X	X	X
BouncePlat	X	X	X	X
DamageDrop	X	X		
HealDrop	X	X		