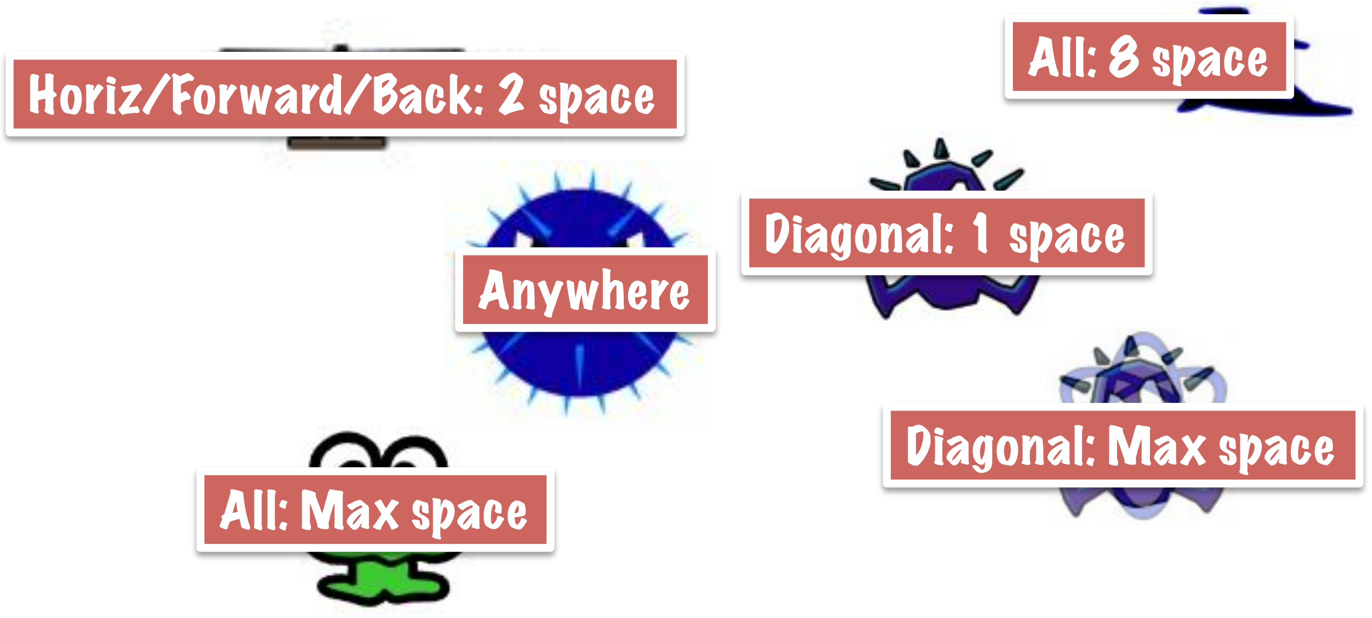
Game Characters



Game Characters

Horiz/

Wing-Back class

: 2 space

All

Kobes class

Angry Blue Spike class

Di

Ghoul class

Dia

nce

Atomic Ghoul class

e

Chopo class

Game Characters

State:
Position
Color to indicate team
Captured?
Life

Behavior:
Move

Wing-Back class

Kobes class

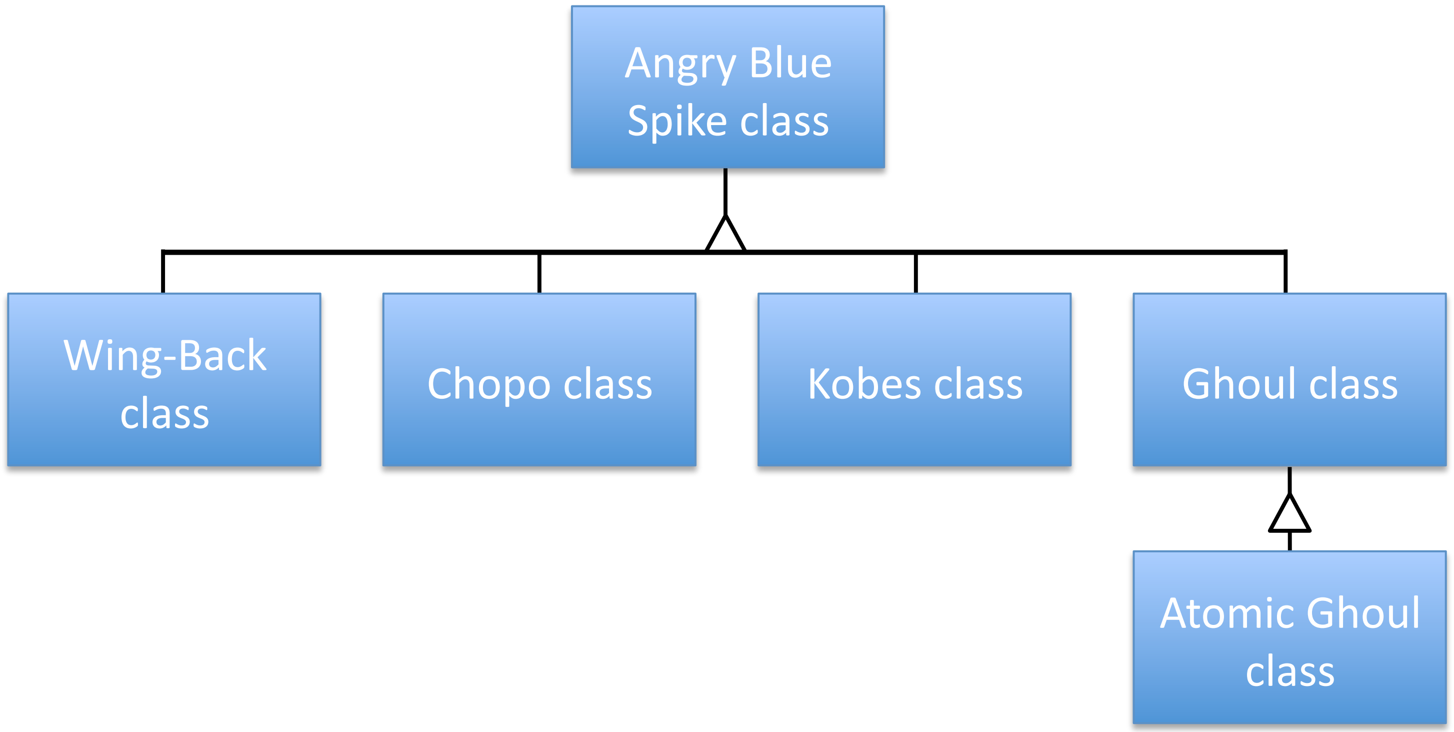
Angry Blue Spike class

Ghoul class

Chopo class

Atomic Ghoul class

Game Characters



Game Characters

- The Angry Blue Spike Class (or ABSpike)
  - simplest or super class
  - all other characters *descended* from this class
- Common fields defined in ABSpike
- Common behaviors defined in ABSpike

subclass

- `class ABSpike` is called the *super class.*
  - Also known as the parent class; in other languages a base class

- `class Kobe` is a *subclass* of `class ABSpike`

- `class Wing-Back` is a *subclass* of `class ABSpike`
  - Also called child classes; in other languages derived class

ABSpike class

- fields common to all of the characters in the game
  - char color
  - boolean captured
  - int life
  - int[] position

ABSpike class

```
public class ABSpike{
   private char color; //R for Red, B for Black
   private boolean captured;
   private int life; //default is 10
   private int[] position = {0,0};
```

ABSpike class

- a constructor that sets
  - color: input parameter
  - captured: will be set to default of false
  - life: set default value to 10
  - position: input parameter

ABSpike class

```
public ABSpike(char team, int[] initPos){
    color = team;
    position[0] = initPos[0];
    position[1] = initPos[1];
    life = 10;
}
```

ABSpike class: set Methods

- Public
  - setColor(char)
  - capture()
- Private
  - setPosition(int, int)
  - setLife(int)

ABSpike class: set Methods

```
public void setColor(char color){
    this.color = color;
}

public void capture(){
    captured = true;
}

private void setLife(int change){
    life = life + change;
}

private void setPosition(int x, int y){
    position[0] = x;
    position[1] = y;
}
```

ABSpike class: get Methods

```
public char getColor(){
   return color;
}

public boolean
   isCaptured(){
   setLife(-1);
   return captured;
}
```

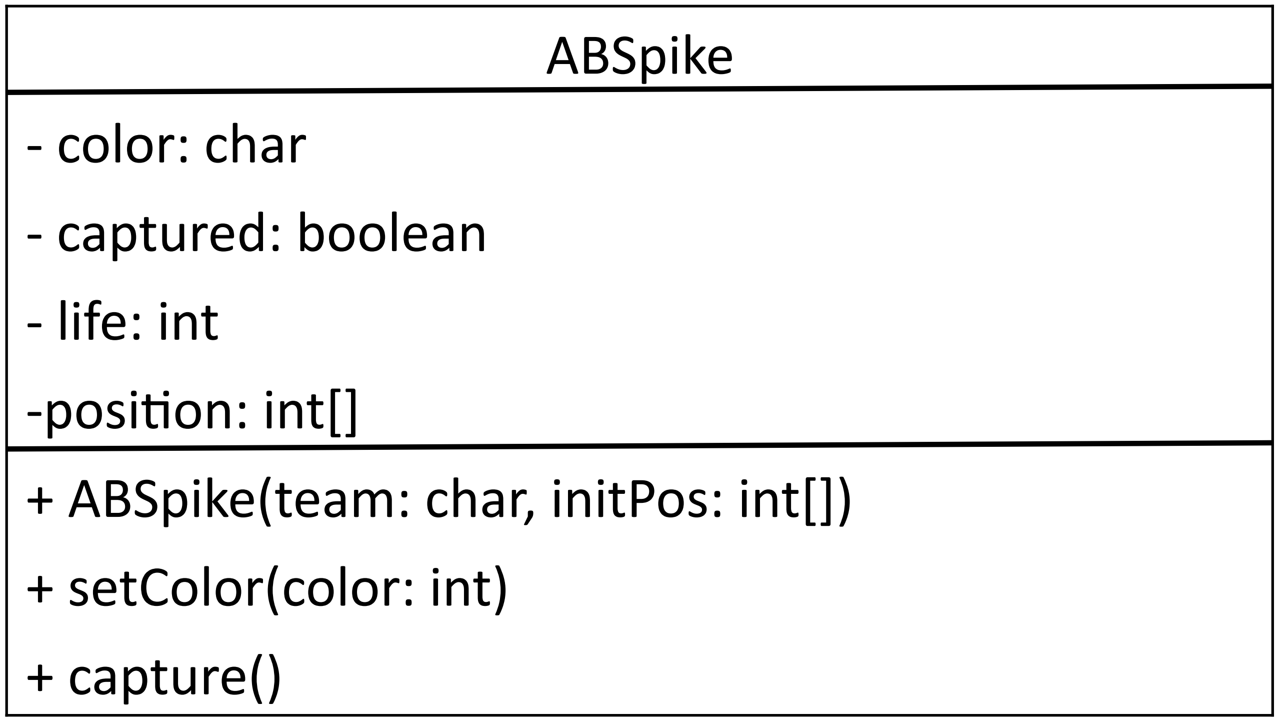
```
public int getLife(){
   return life;
}

public int getX(){
   return position[0];
}

public int getY(){
   return position[1];
}
```

ABSpike class: other mutators

```
public boolean move (int x, int y){
  if (life>0){
      setLife(-1);
      setPosition(x,y);
      return (true);
  }

  else return(false);
}
```

Override the toString() method

the ABSpike class



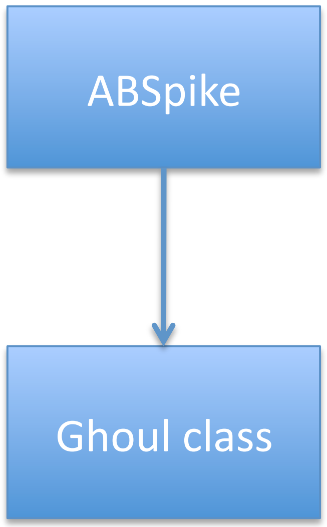
continued…

the ABSpike class

...continued

```
- setLife(change: int);
- setPosition(x: int, y: int);
+ getColor(): char
+isCaptured(): boolean
+ getX(): int
+ getY(): int
+ move(x: int, y: int): boolean
+ getLife(): int
```

the Ghoul class



subclass

```
public class Derived_Class_Name extends super_Class_Name
{
   Declarations of additional fields

   Definitions of additional methods
   and overridden methods
}
```

subclass

- Declare only the **added fields** and define only the **new and overridden** methods.

- The variables and methods of the super class which are inherited automatically
  - all `Ghoul` objects are also `ABSpike` objects, and thus they include color, captured, life and position fields
  - we can call the public methods of the ABSpike class on Ghouls as well

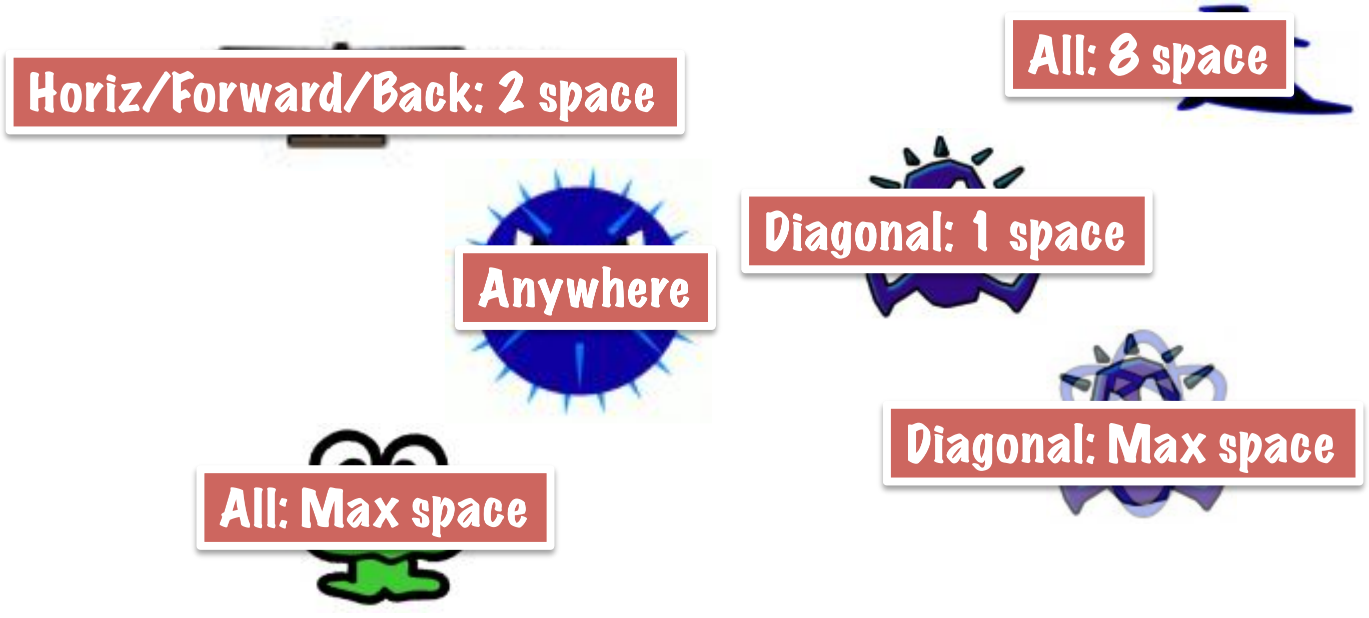
Ghoul class

```
public class Ghoul extends ABSpike{

    public Ghoul(char team, int[] startingPos){
        super(team, startingPos);
        setLife(5); //life set to default + 5
    }

    public boolean move (int x, int y){
        if(1 == Math.abs(getX() - x) && 1 == Math.abs(getY() - y)){
            return(super.move(x, y));
        }
        return(false);
    }
}
```

Creating a Ghoul

```
Ghoul Jim = new Ghoul('R', position);
Jim.capture();
Jim.move(1,1);
```

Game Characters



ABSpike class move method

```
public boolean move (int x, int y){
  if (life>0){
      setLife(-1);
      setPosition(x,y);
      return (true);
  }

  else return(false);
}
```

override this definition

Ghoul class move method

- Move the Ghoul if
  - the proposed location is one space in either diagonal direction

    |x – newX|==1 and |y – newY|==1
  - AND life > 0
- Else return false

move(x,y)

ABSpike

- life >0
  - life = life – 1
  - set new position
  - return true
- life <=0
  - return false

Ghoul

- legal move AND life >0
  - life = life – 1
  - set new position
  - return true
- life <=0
  - return false
- illegal move
  - return false

move(x,y)

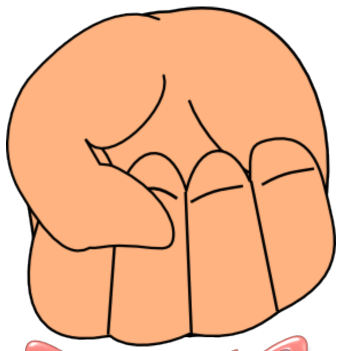
ABSpike

- life >0
    - life = life – 1
    - set new position
    - return true
- life <=0
    - return false

Ghoul

- legal move
    - life >0
        - life = life – 1
        - set new position
        - return true
    - life <=0
        - return false
- illegal move
    - return false

Why skimp on



Code Writing?

- only write and test a method once
- minimize errors
- easier to debug
- changes and updates are simpler
- code may be easier to read

move(x,y)

ABSpike

- life >0
  - life = life – 1
  - set new position
  - return true
- life <=0
  - return false

Ghoul

- legal move
  - life >0
    - life = life – 1
    - set new position
    - return true
  - life <=0
    - return false
- illegal move
  - return false

move(x,y)

Ghoul

- legal move

  Call `move(x,y)` *in Character class*

- illegal move
  - return false

Ghoul class move method

```
public boolean move (int x, int y){
   if(1 == Math.abs(getX() - x)
         && 1 == Math.abs(getY() - y)){
      return(super.move(x, y));
   }
   return(false);
}
```

The move method

```
public boolean move (int x, int y){
  if (life>0){
    setLif

  }

  el
}
```

```
public boolean move (int x, int y){
   if(1 == Math.abs(getX() - x)
         && 1 == Math.abs(getY() - y)){
      return(super.move(x, y));
   }
   return(false);
}
```