

Classes -- Inheritance

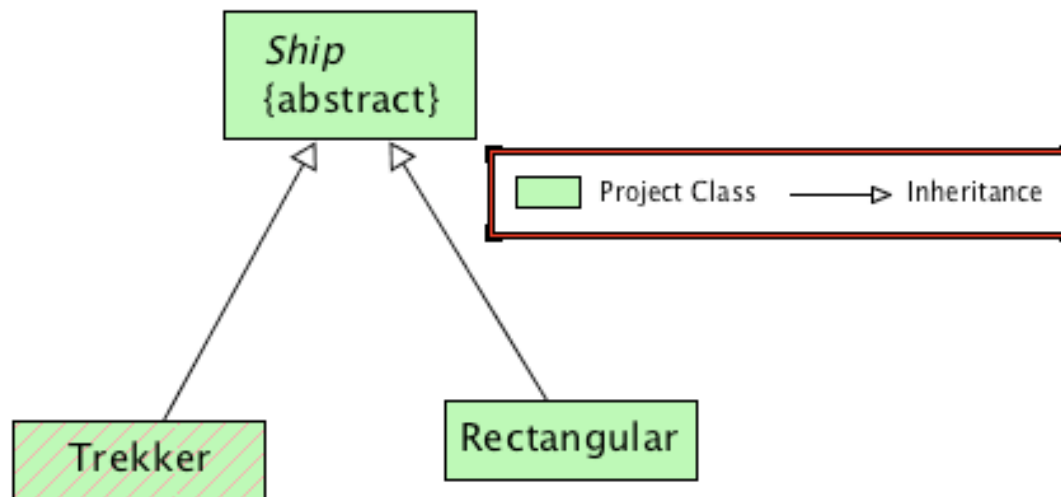
Atul Prakash

Supporting Multiple Ships

- Suppose there are different models of ships in our game -- different shapes:
 - Rectangular
 - Trekker
 - SharpShooter
 - SimpleShip

Inheritance

- One class per ship type. Indicate that they are all Ships.



Methods common to all ships

- We want standard features on all ships:
 - Print the shape of the ship
 - Tell a ship its location on the grid
 - Tell a ship the location of a shot
- Check if a ship is destroyed
- Features are independent of the shape of the ship

Define a Parent Ship Class

```
public class Ship {
    private String name;

    public Ship(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    // Specific types of ships must implement these methods.
    public void printShip() { ??? };
}
```

Every ship must know how to print itself,
but how does a generic ship print itself?

Abstract methods

- We want all ships to provide a method `printShip()` that prints the shape of the ship.
- But, the method cannot be implemented by the generic Ship class -- it does not know the shape.
- In Java, you indicate that by declaring the method to be **abstract**.
- If a class has abstract methods, then the class is also declared to be **abstract**.

```

public abstract class Ship {
    private String name;
    protected int posX; // x-location on the grid
    protected int posY; // y-location on the grid
    public Ship(String name) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    // Specific types of ships must implement these methods.
    public abstract void printShip();
    public abstract boolean setLocation(int x, int y, int xbound, int
ybound); // location on the grid.

    // Returns true if ship is hit, else false.
    public abstract boolean shotAt(int x, int y); // x, y are coordinates
relative to the grid.

    public abstract boolean isDestroyed(); // true if ship destroyed
}

```

Non-abstract methods have an implementation.
Abstract methods do not. A subclass must implement the
method.

Creating a Sample Ship Class

```
public class SimpleShip extends Ship {
```

```
    boolean[][] shape;
```

```
    public SimpleShip(String name) {  
        super(name);  
        shape = new boolean[4][2];  
        for (int i = 0; i < 4; i++) {  
            shape[i] = new boolean[2];  
        }  
        for (int i = 0; i < 4; i++) {  
            for (int j = 0; j < 2; j++) {  
                shape[i][j] = true;  
            }  
        }  
    }  
}
```

Call the parent class's initializer

Initialization specific to
SimpleShip

Ship Shape: xxxx

xxxxx

Other Methods in SimpleShip

```
@Override
public void printShip() {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 2; j++) {
            System.out.print('x');
        }
        System.out.println();
    }
}
@Override
public boolean isDestroyed() {
    ...
}
@Override
public boolean setLocation(int x, int y, int xbound, int ybound) {
    ...
}
@Override
public boolean shotAt(int x, int y) {
    ...
}
```

Protected Members

- We saw public and private variables and methods.
- **protected** members:
 - Visible to the subclasses and to the classes in the same directory (package), but not to the rest of the world