From the Soapbox

- What is an interface?
- Can you make a video about interfaces?

© Peter Andreae and Karsten Lundqvist

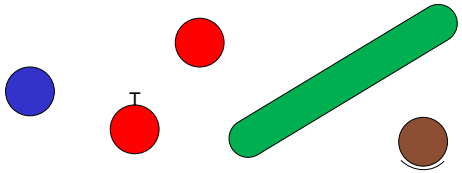
BalloonGame

- From COMP102, the Balloon game
  - Collection of Balloons
  - Clicking on a Balloon expands it
  - If two Balloons touch, they pop.
  - Score = total area of live balloons minus area of popped Balloons.
- Key data structure:
  - Needs a list of Balloon objects:
    - ArrayList <Balloon>

© Peter Andreae and Karsten Lundqvist

Collections of different types

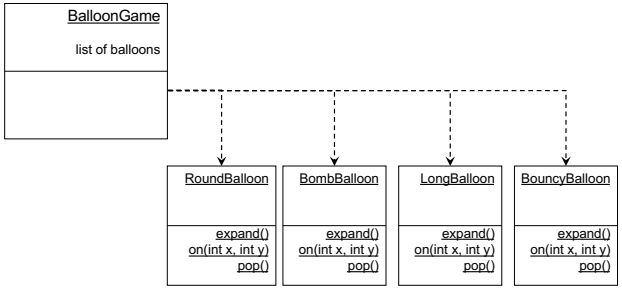
- Extending the Balloon Game:
  - several kinds of balloons:
    - ordinary balloons – expand uniformly, and pop on touching
    - bomb balloons – blow up, taking out lots of balloons at max size
    - long balloons – expand into sausage shapes, popping other balloons only, up to limit
    - bouncy balloons – don't expand, but bounce around and move other balloons



© Peter Andreae and Karsten Lundqvist

New Balloon Game

- BalloonGame needs a list to hold all the balloons



- Each class of balloons needs the same methods:
  - expand, on, pop
  - but, they each behave differently.

© Peter Andreae and Karsten Lundqvist

Different Balloon classes for a game

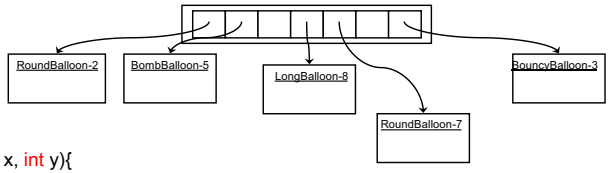
```
public class RoundBalloon {
    private int x, y;
    private int radius = 10;
    private Color col;
    public void expand(int amt){…..
    public boolean on(int x, int y){…..
}
public class LongBalloon {
    private int x, y;
    private int length = 10;
    private double direction = 1.26;
    private Color baseCol, tubeCol;
    public void expand(int amt){…..
    public boolean on(int x, int y){…..
}
```

© Peter Andreae and Karsten Lundqvist

BalloonGame

- Needs to store all balloons in the game. One way to do this

```
public class BalloonGame {
    :
    private int maxBalloons = 20;
    private List<Object>  balloons =  new ArrayList<Object>();
```




```
    public void doStep(int x, int y){
        for (int i = 0; i<balloons.size(); i++){
            if  (this.balloons.get(i).on(x, y) ) {
                this.balloons.get(i).expand();
                :
```

Java will complain!
No such method for Object

© Peter Andreae and Karsten Lundqvist

Casting

- Can sometimes convert a value from one type to another
  - numbers:

    ```
    int col =  (int)  (x /cellWidth);
    double x  =  100 * col;
    ```

    - Must be an explicit cast if information might be lost.
  - anything to a string: (by adding to a string)

    ```
    "value is " + x
    ```

    - For an Object value, it automatically calls the toString() method

    ```
    "value is " + x.toString()
    ```

  - any object value from one of its types to another

    ```
    Object  balloon =  (Object)  ( new  RoundBalloon(34, 45)  );
    RoundBalloon rb = (RoundBalloon) balloon;
    ```

    - object must be of that type, otherwise a runtime error

© Peter Andreae and Karsten Lundqvist

A Balloon type

- RoundBalloon, LongBalloon, BombBalloon, BouncyBalloon are all kinds of Balloon.

  ⇒ should all be of a Balloon type.

  RoundBalloon-2 should be a
  - a RoundBalloon,
  - a Balloon, and
  - an Object

```
public class BalloonGame {
    private int maxBalloons = 20;
    private  List<Balloon> balloons =  new   ArrayList<Balloon>();

    public void doStep(int x, int y){
        for (int i = 0; i<balloons.size(); i++){
            if  (this.balloons.get(i).on(x, y) ) {
                this.balloons.get(i).expand();
```

© Peter Andreae and Karsten Lundqvist

Making a Balloon Type

- Problem: What is "a Balloon" ?
- Answer 1:
  - Something you can
    - expand, `public void expand()`
    - ask if a point is on it, `public boolean on(int x, int y)`
    - pop `public void pop()`
    - ask for its size `public double size()`
    - ask if touching a balloon `public boolean touches(Balloon other)`
- Answer 2:
  - a RoundBalloon, or
  - a LongBalloon, or
  - a BombBalloon, or
  - a BouncyBalloon.
- We need to make both answers true.

© Peter Andreae and Karsten Lundqvist

Interfaces

- An interface describes a supertype of other types.
  - It specifies
    - a name for the type
    - the headers for the methods that any object of the type can respond to.
    - constants (`public static final double` GRAVITY = 9.8)
    - **nothing else!** (no fields, constructors, or method bodies)

```
public interface Balloon {
    method headers
    constants
}
```

Latest version of Java has extensions

- You cannot make a new instance of an interface.

```
Balloon b = new Balloon();
```

An interface just says "what"; not "how"

© Peter Andreae and Karsten Lundqvist

A Balloon Interface.

```
public interface Balloon {
    public void expand();
    public boolean on(int x, int y);
    public void pop();
    public double size();
    public boolean touches(Balloon other);
}
```

Note the ; instead of { …. }

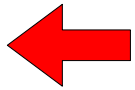
- Declares a type that you can use for fields, variables, arrays.

© Peter Andreae and Karsten Lundqvist

Making a Balloon Type

- Problem: What is "a Balloon" ?
- Answer 1: Defined by `interface Balloon {`
  - Something you can
    - expand,
    - ask if a point is on it,
    - pop
    - ask for its size
    - ask if touching a balloon

- Answer 2:
  - a RoundBalloon, or
  - a LongBalloon, or
  - a BombBalloon, or
  - a BouncyBalloon.



- We need to make both answers true.

© Peter Andreae and Karsten Lundqvist

Making classes have another type

- If you define some class,
  - all instances are automatically of type Object also.
- To make instances be of some interface type:
  - declare that the class implements the interface type:
  - make sure that the class defines all the methods specified by the interface type:

implements is a "promise" that these objects will be of the interface type. ie, will have all the methods.

```
public class RoundBalloon implements Balloon {
    private int x, y;              private int radius = 10;
    private Color col;
    public void expand(int amt){ …… }
    public boolean on(int x, int y){ …… }
    public void pop() { …… }
    public double size() { …… }
    public boolean touches(Balloon other) { …… }
}
```

Must provide method bodies, not just headers!

© Peter Andreae and Karsten Lundqvist

More classes implementing Balloon

```
public class LongBalloon implements Balloon {
    private int x, y;              private int length = 10;
    private double dir = 1.26;
    private Color baseCol, tubeCol;
    public void expand(int amt){ …… }
    public boolean on(int x, int y){ …… }
    public void pop() { …… }
    public double size() { …… }
    public boolean touches(Balloon other) { …… }
}
public class BombBalloon implements Balloon {
    private int x, y, maxRad;      private int radius = 10;
    private Color col;
    public void expand(int amt){ …… }
    public boolean on(int x, int y){ …… }
    public void pop() { …… }
    public double size() { …… }
    public boolean touches(Balloon other) { …… }
}
```

© Peter Andreae and Karsten Lundqvist

Making a Balloon Type

- Problem: What is "a Balloon" ?
- Answer 1: Defined by **interface** Balloon {
  - Something you can
    - expand,
    - ask if a point is on it,
    - pop
    - ask for its size
    - ask if touching a balloon

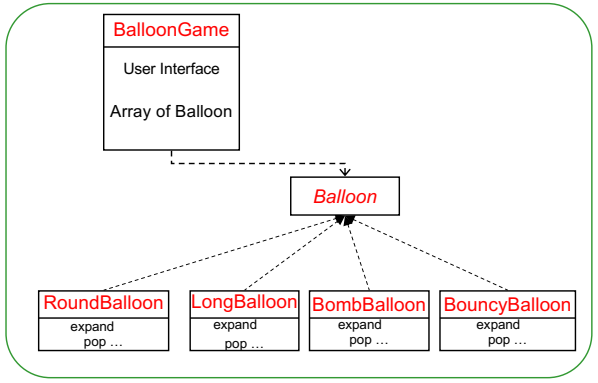
- Answer 2: Specified by … **implements** Balloon
  - a RoundBalloon, or
  - a LongBalloon, or
  - a BombBalloon, or
  - a BouncyBalloon.

- We have now made both answers true.

© Peter Andreae and Karsten Lundqvist

BalloonGame class structure

-



© Peter Andreae and Karsten Lundqvist

Interface summary

- If you define an interface:
  - You have defined a type
  - You ***can*** declare variables (or arrays) to have that type.
  - You ***cannot*** make an object of the interface

    `new Balloon(…)  // NOT ALLOWED (there is no constructor)`
  - Objects from classes **implement**ing the interface are of that type.
  - You can call any of the specified methods on values in fields/variables of that type.
- When defining an interface:
  - You should include headers of methods
  - You may include **static final** constants
  - You may ***not*** include fields, constructors, or method bodies.

© Peter Andreae and Karsten Lundqvist

More about the Collections library

- Java provides more methods for the collections than just the "essential" operations on the collections.
- Makes them more useful and easier to use.
- Quick run through some of the documentation
- Note: Stack class predated the Collections library, and doesn't have an ADT interface.

© Peter Andreae and Karsten Lundqvist

More operations on Collections

- Collection:
  - isEmpty(), size(), clear(),
  - **add(…), remove(…), contains**(…)
  - addAll(…), removeAll(…), containsAll(…), retainAll(…), removeIf(…)
  - toArray()
  - for(E item : collection){….}
- Set: collection that won't allow duplicates
  - exactly the operations for Collection
  - HashSet, TreeSet
- List:
  - additional operations based on the order:
  - add(index, …), get(index), remove(index), indexOf(…), set(index, …), sublist(…)
  - ArrayList, LinkedList

© Peter Andreae and Karsten Lundqvist

Useful Operations on Collections

- Collections class
  - all static methods: Collections.sort(*list*);
  - min(…), max(…), frequency(…),
  - fill(…), reverse(…), shuffle(…), sort(…), swap(…)
  - …
- Also, see the Arrays class.

© Peter Andreae and Karsten Lundqvist

Using Sets

- Vocabulary:
  - Given a file of words (from a book)
  - Count the number of words and the number of distinct words.
  - Is using a Set faster than using a List?
    - why?

© Peter Andreae and Karsten Lundqvist

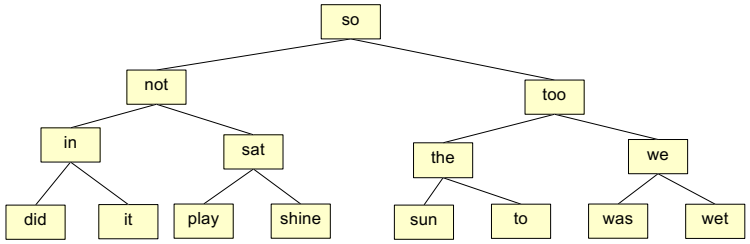
Sets: HashSet

- HashSets:
  - uses an array to store the values.
  - given a value, compute an index where it belongs (hashCode)
  - jump to that index in the array
  - speed is independent of how big the set is!!!
  - Problem: suppose two values have the same hashCode?
    - make a "bucket" – a list of values, and search down the list
    - OK, as long as the HashSet doesn't get too full
    - Of the HashSet gets a bit full (eg, 70%)
      - make a new array (double the size) and move all the values over
  - Problem: order of items is all mixed up

© Peter Andreae and Karsten Lundqvist

TreeSet

- TreeSets:
  - Keep all the values in a tree structure, alphabetically organised.
  - Search down the branches to find values
  - Not quite as fast as HashSets, but very close!
  - Million items – only 20 steps to get to any item.



© Peter Andreae and Karsten Lundqvist

Using Sets

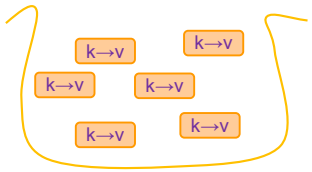
- Vocabulary:
  - Given a file of words (from a book)
  - Count the number of words and the number of distinct words.
  - Is using a Set faster than using a List?
    - why?
  - What about printing out the vocabulary?
    - which order?
      - alphabetical order?
      - order they first occurred in the book?
      - frequency?
  - Using a Map for keeping track of frequency.

© Peter Andreae and Karsten Lundqvist

Maps

A Map is

- a collection of data with an index:
  - index → data
- a collection of key → value pairs
- a mapping from keys to values



- eg: dictionary: word → definition
- eg: phonebook: name → phone number
- eg: Bank: account number → account

- Note: the keys must be unique (the keys are a Set); values can be duplicated

© Peter Andreae and Karsten Lundqvist

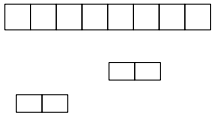
Maps in Java

- Map is part of the Collection library
- To declare a Map, need two types: key and value
  - `private Map<String, String> dictionary = new HashMap<String, String>();`
    - *given a word, get the definition*
  - `private Map<String, Person> members = new HashMap<String, Person>();`
    - given a name, get the Person object for that person.
- Fundamental operations:
  - get(key)
  - put(key, value)
  - remove(key)
- Implementations: HashMap, TreeMap (like HashSet and TreeSet)

© Peter Andreae and Karsten Lundqvist

Maps in Java:

- Map is a collection of key-value pairs:
  - Type of pairs: Map.Entry<String,Person>
  - organised by the key (fast to access, given a key)
- operations:
  - get(key) put(key, value)
  - remove(key) replace(key, newvalue)
  - clear(), isEmpty(), size()
  - containsKey(key), containsValue(value)
  - keySet() -> Set<keytype>
  - values() -> Collection<valuetype>
  - entrySet() -> Set<Map.Entry<keytype, valuetype>>



© Peter Andreae and Karsten Lundqvist