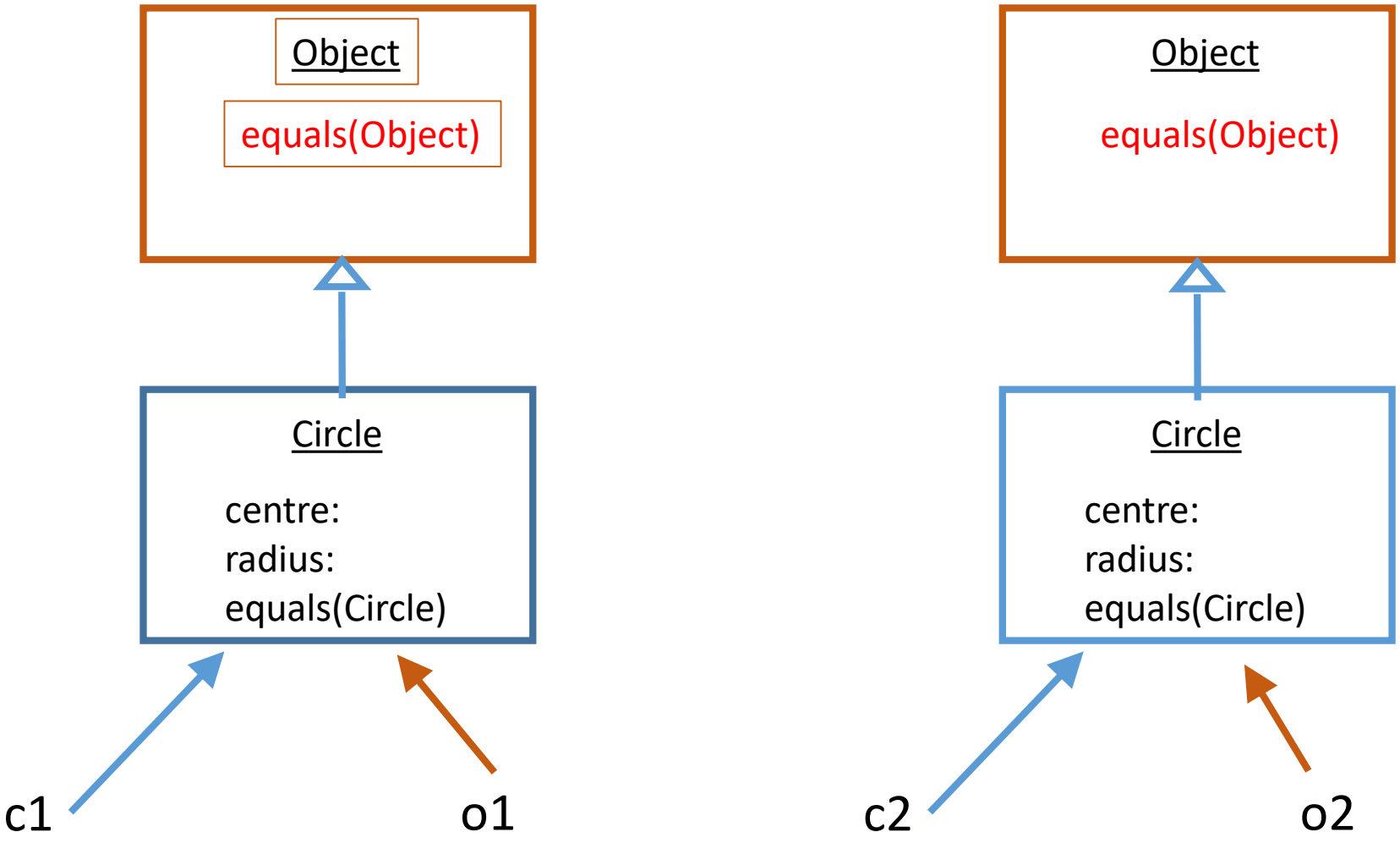
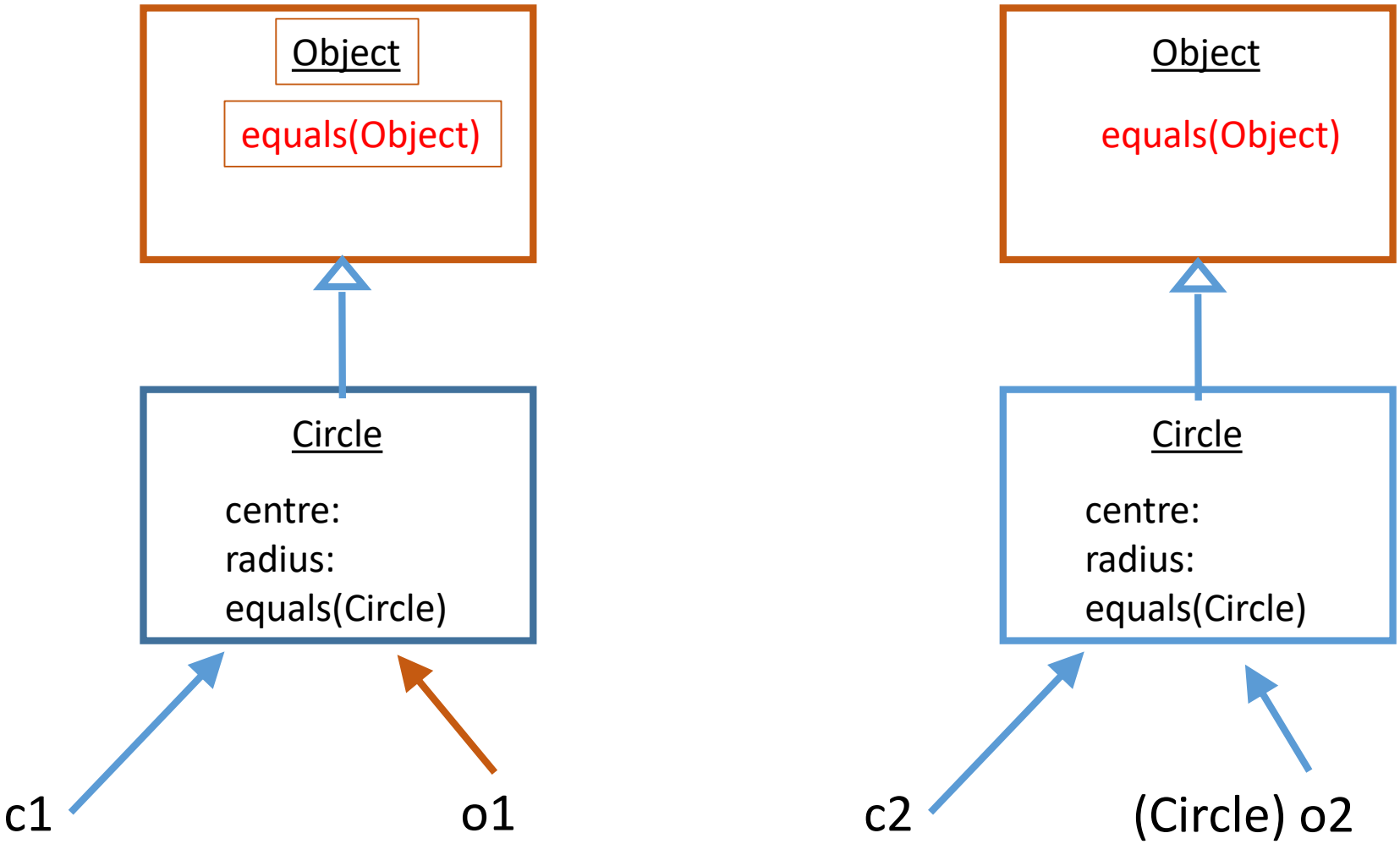
Recitation 2

Q1a,c,d,e,g,h


Object
equals(Object)
Circle
centre:
radius:
equals(Circle)
c1
o1
Object
equals(Object)
Circle
centre:
radius:
equals(Circle)
c2
o2

Q1b,f


Object
equals(Object)
Circle
centre:
radius:
equals(Circle)
c1
o1
Object
equals(Object)
Circle
centre:
radius:
equals(Circle)
c2
(Circle) o2

Q2(c)

Liskov Substitution Principle (LSP):

> The client of a class *T* expects the same behaviour if *T* was substituted with a subclass *S* of *T*

LSP is a runtime behavior. It cannot be checked by a compiler.

Q2(c)

- The Rectangle class has these contracts:
  - The **`height`** can only be set by the Constructor or the **`setHeight()`** method, and nothing else.
  - The **`width`** can only be set by the Constructor or the **`setWidth()`** method, and nothing else.

- The client (user) of **`Rectangle`** expects these contracts to be fulfilled by **`Rectangle`** and all its subclasses.
  - But **`Square`** doesn't, since **`height`** can be changed by **`setWidth()`**

- Therefore, **`Square`** violates LSP!
- Can we make **`Rectangle`** the subclass of **`Square`** instead?
- How else can we solve this problem?

Rules for overriding Java methods

1. Only inherited methods can be overridden.

   Superclass methods that are **`public`**, **`protected`**, or **`default`** can be overridden. **`private`** methods cannot.

2. Final and static methods cannot be overridden.

   Superclass methods declared as **`final`** or **`static`** cannot be overridden.

Rules for overriding Java methods

3. The overriding method (subclass) must have the same argument list as the overridden method (superclass).

   Otherwise the subclass method is overloading, not overriding.

   Use the `@Override` declaration to trigger compiler check.

Rules for overriding Java methods

4. The overriding method must return the same type, or subtype.

```
//Suppose D is a subclass of C
class A { C foo() … }
class B {
   @Override
   C foo() … // ok
   D foo() … // also ok
   String foo() … // not ok
}
```

Rules for overriding Java methods

5. The overriding method must not have a more restrictive access modifier.

6. Abstract methods must be overridden by concrete subclass.

More rules here:

https://www.codejava.net/java-core/the-java-language/12-rules-of-overriding-in-java-you-should-know

Abstract class vs Interface

- Generally speaking, use Interface if you don't have any attributes, AND all your methods are deferred (ie specified now, but implemented later).
  - Usually, name your Interface "able", to specify that this Interface provides the ability to do something.
  - Examples: Comparable, Scalable, Flexible

- Use Abstract class if you have attributes, or want to implement some methods while deferring others.
  - Usually, class names should be nouns and not verbs or adjectives

- https://www.geeksforgeeks.org/difference-between-abstract-class-and-interface-in-java/

Sorting example

- Suppose you wish to sort an array of Things.
- Thing could be any class, eg String, Point, Circle, StudentRecord, Customer, etc.

```
void sort(Thing[] arr) { …
   if (arr[i] <= arr[j]) …
}
```

- The ability to sort hinges on the property of "less than or equal to". It must be possible for one Thing to be "less than or equal to" another Thing.
  - In CS1231 lingo: you need a partial order
- But the details of "less than or equal to" depends on the actual type, which **`sort`** shouldn't care about.

```
interface Sortable {
    boolean lessThanEQ(Sortable y);
}

class C implements Sortable {
    private int x;
    private String color = “red”;
    public C(int a) { this.x = a; }

    @Override
    public boolean lessThanEQ(Sortable other) {
        return this.x <= ((C) other).x;
    }
}
```

```
void sort (Sortable[] arr) {
    for (int i=0; i<arr.length-1; i++)
        if (arr[i].lessThanEQ(arr[i+1]))
            System.out.println("in place");
        else System.out.println("out of place");
}


C[] cArray = {new C(5), new C(-1), new C(30), new
C(40), new C(15) };
sort(cArray);
```

```
out of place
in place
in place
out of place
```