برنامه نویسی پیشرفته
C#

۱۴ مهر ۹۸

ملکی مجد

مباحث

- نوشتن برنامه برای درک تعریف کلاس ، ساخت شی و استفاده از شی

- نکته مربوط به Static
- var
- null
- Boxing
- Unboxing
- is and as

Write a sample code

- Define class point
- New an object from point
- Overload constructor (default not available)
- Write default constructor
- Call constructors
- Public and private Data fields
- Public and private methods
- Public method use private data (and method)
- Static data and method (count the objects)
- Distance between two points

Static fields

- Static method and data
  - Only accessed by class name (not instance)

var

- the *var* keyword causes the compiler to create a variable of the same type as the expression used to initialize it.

- Var i = 42;
- Var point = new point();

Copying value type variables and classes

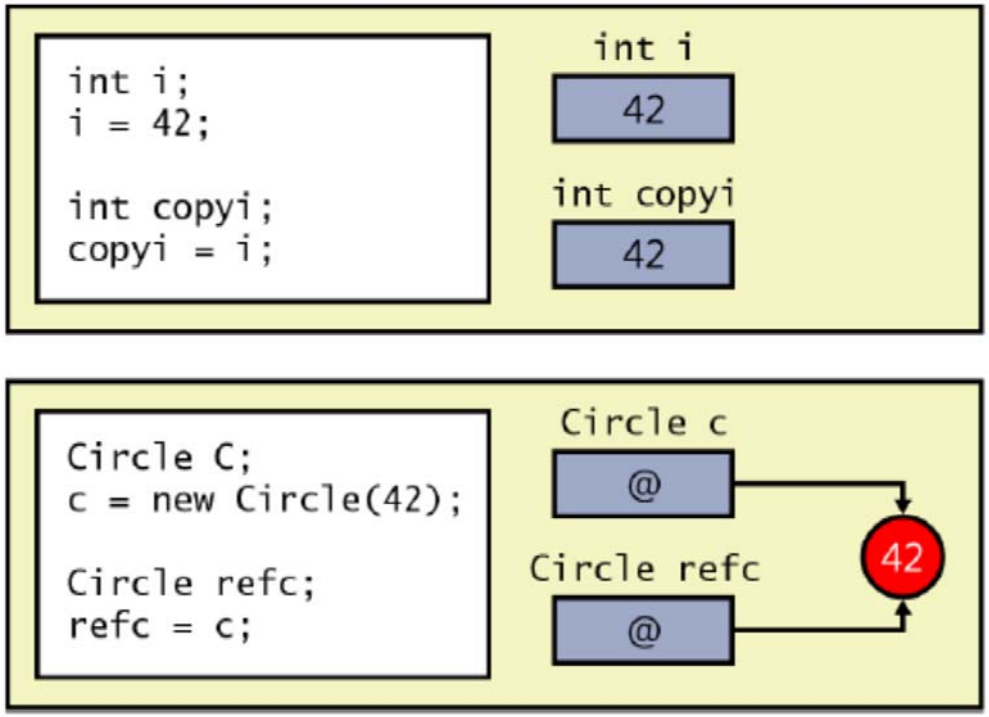
- *value types*
  - *int*, *float*, *double*, and *char*
  - have a fixed size, and when you declare a variable as a value type, the compiler generates code that allocates a block of memory big enough to hold a corresponding value

- Class types
  - Point
  - is allot a small piece of memory that can potentially hold the address of (or a reference to) another block of memory containing a *Point.* (An address specifies the location of an item in memory.)
  - The memory for the actual *Point* object is allocated only when the *new* keyword is used to create the object.
  - A class is an example of a *reference type*

example

```
int i = 42;
int copyi = i;
i++;
```

```
Circle c = new Circle(42);
Circle refc = c;
```



Copying reference types and data privacy

- to copy the contents of a *Circle* object, *c*, into a different *Circle* object, *refc*, instead of just copying the reference, you must make *refc* refer to a new instance of the *Circle* class and then copy the data, field by field, from *c* into *refc*

- Private filed ?
- a class could provide a *Clone* method that returns another instance of the same class but populated with the same data

Clone method for the *Circle* class

```
class Circle
{
    private int radius;
    // Constructors and other methods omitted
    ...
    public Circle Clone()
    {
        // Create a new Circle object
        Circle clone = new Circle();

        // Copy private data from this to clone
        clone.radius = this.radius;

        // Return the new Circle object containing the copied data
        return clone;
    }
}
```

deep copy and shallow copy

- one or more fields are themselves reference types
  - reference types also need to provide a *Clone* method

- Shallow copy
  - the *Clone* method simply copies references
- Deep copy
  - the *Clone* method is used for fields that are reference types

Null value

- In C#, you can assign the *null* value to any reference variable
  - The *null* value simply means that the variable does not refer to an object in memory

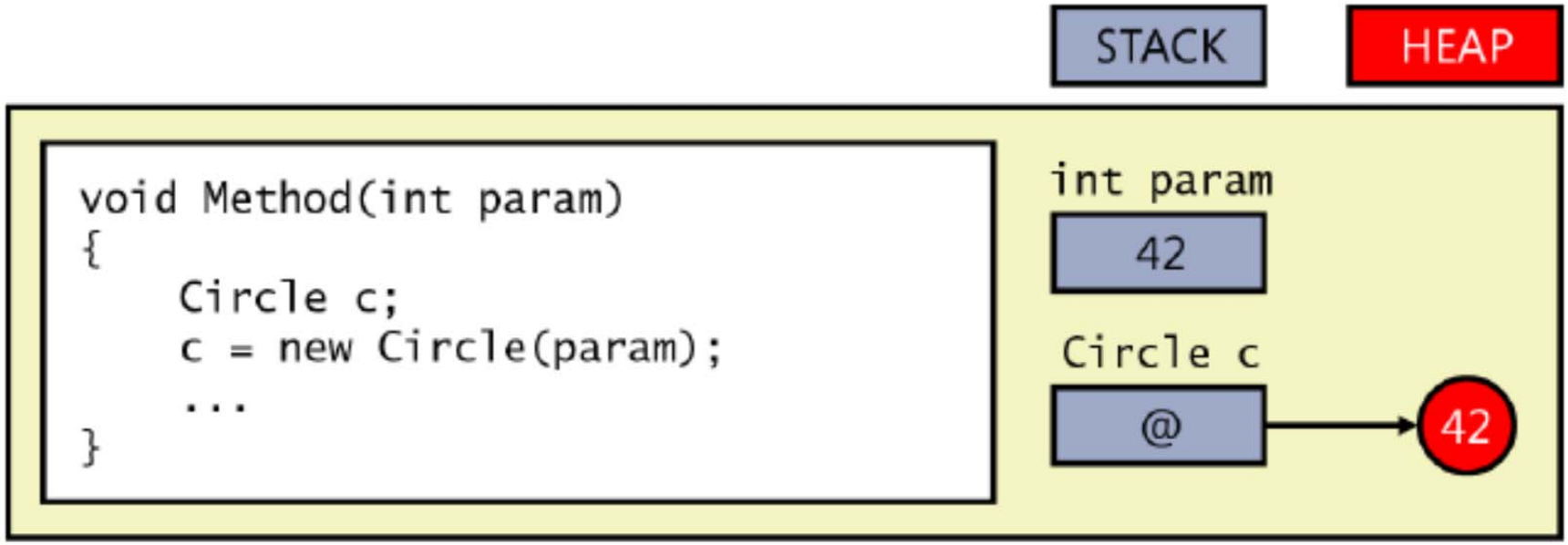
```
Circle c = new Circle(42);
Circle copy;                // Uninitialized !!!
...
if (copy == // only assign to copy if it is uninitialized, but what goes here?)
{
    copy = c;               // copy and c refer to the same object
    ...
}
```

```
Circle c = new Circle(42);
Circle copy = null;         // Initialized
...
if (copy == null)
{
    copy = c;               // copy and c refer to the same object
    ...
}
```

Memory organization (for program execution)

- Stack
  - In a block of code: the memory required for parameters and local variables is always acquired from the stack
  - When finish: released back to the stack
  - Method parameters and local variables on the stack have a well-defined life span: they come into existence when the method starts, and they disappear as soon as the method completes
- Heap
  - the memory required to build the object is always acquired from the heap.
  - same object can be referenced from several places by using reference variables.
  - more indeterminate life span; an object is created by using the *new* keyword, but it disappears only sometime after the last reference to the object is removed

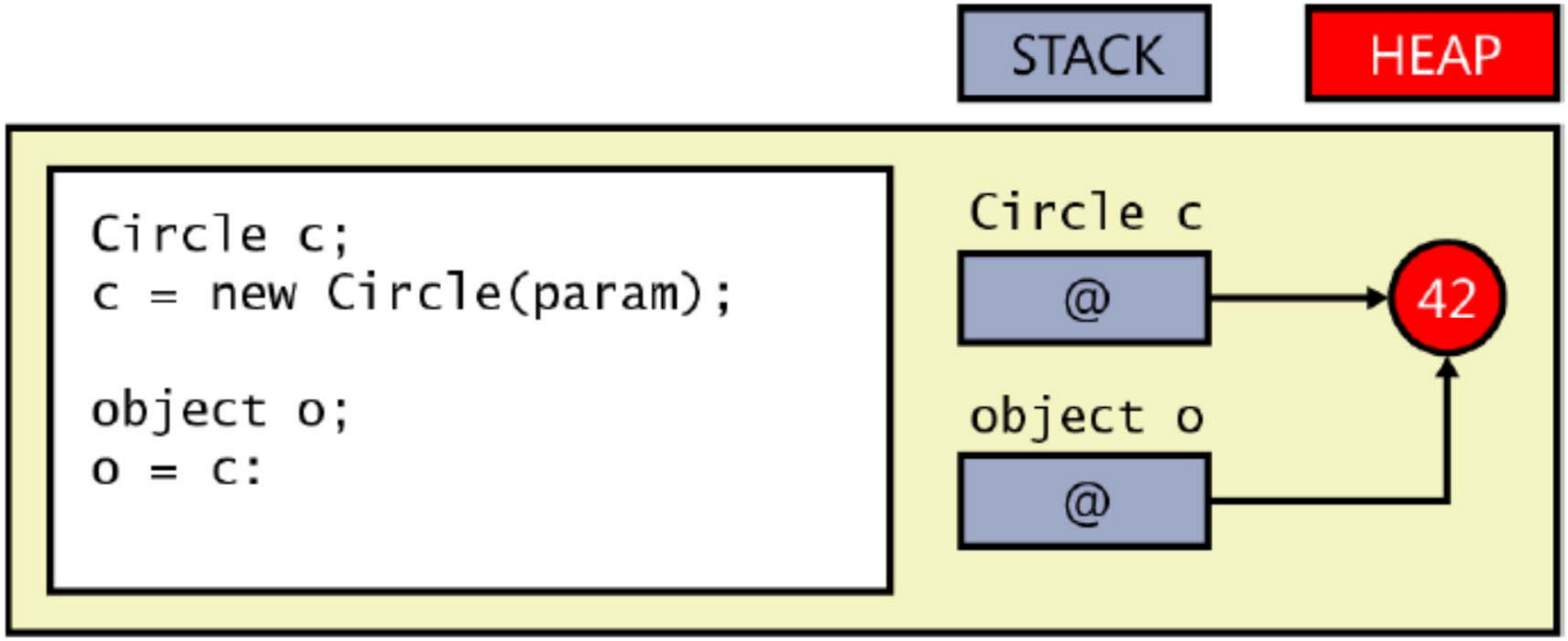
Example of life span and memory organization



The *System.Object* class

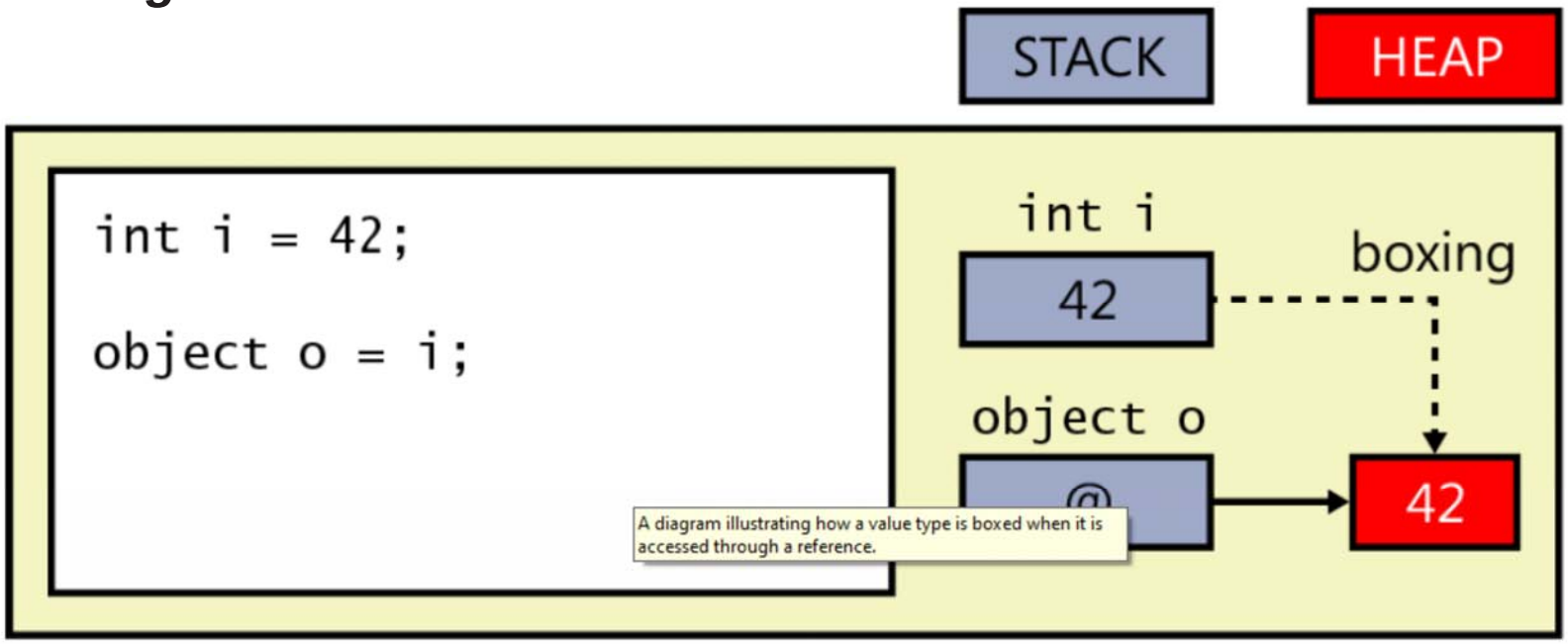
- One of the most important reference types in the .NET Framework is the *Object* class in the *System* namespace
  - all classes are specialized types of *System.Object (inheritance)*
  - you can use *System.Object* to create a variable that can refer to any reference type
  - *object* keyword as an alias for *System.Object*

object


STACK
HEAP
Circle c;
c = new Circle(param);
object o;
o = c:
Circle c
@
42
object o
@

Boxing

automatic copying of an item from the stack to the heap is called *boxing*



Boxing

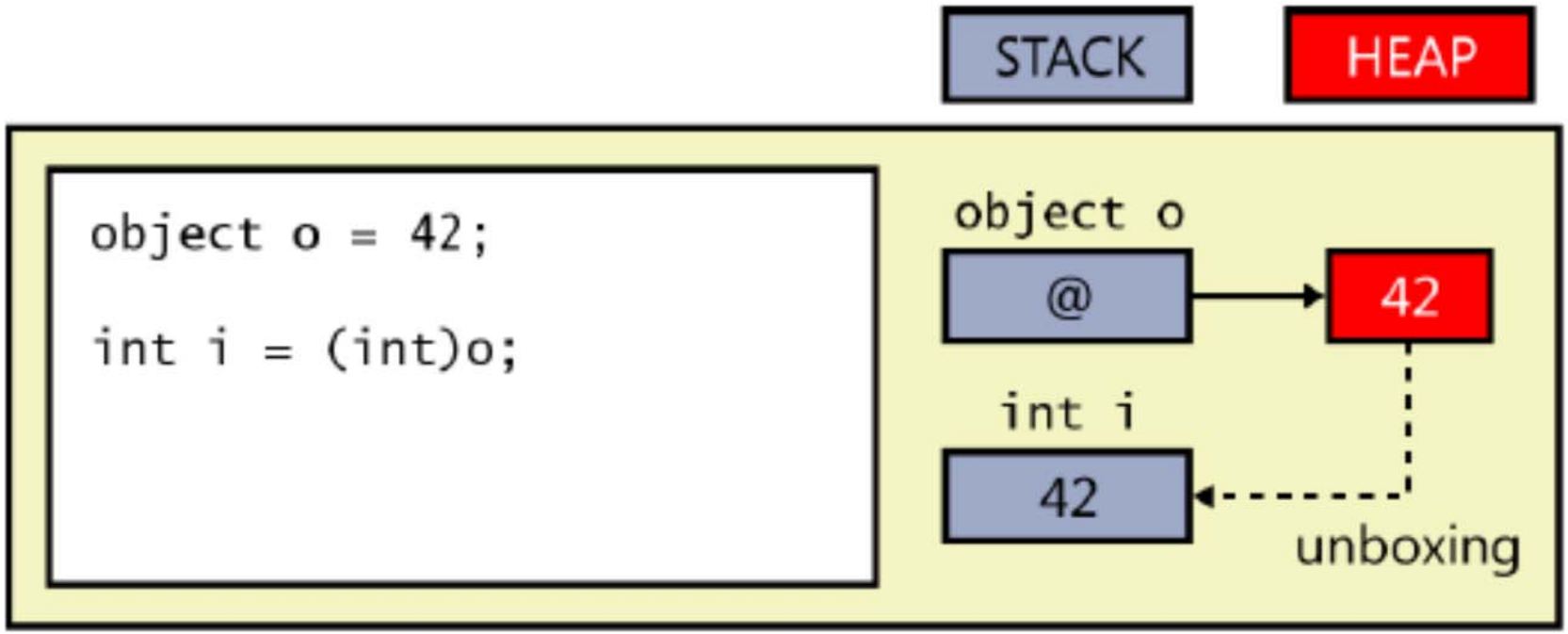
- **Important** If you modify the original value of the variable *i*, the value on the heap referenced through *o* will not change. Likewise, if you modify the value on the heap, the original value of the variable will not change.

Unboxing (must use what is known as a *cast)*

```
int i = o; ×
```

```
int i = 42;
object o = i;  // boxes
i = (int)o;                // compiles okay
```

Valid unboxing



InvalidCastException