Object Stacks

```
class ArrayObjectStack implements ObjectStack {
  Object[] stack;
  int size;
  int N;

  ArrayObjectStack() {
    N = 100;
    stack = new Object[N];
    size = 0;
  }

  public void push(Object x) {
    assert (size < 100);
    stack[size] = x;
    size = size + 1;
  }

  public Object pop() {
    assert (size > 0);
    Object result = stack[size-1];
    size = size - 1;
    return result;
  }

  public Object peek() {
    assert (size > 0);
    Object result = stack[size-1];
    return result;
  }

  public boolean empty() {
    return (size == 0);
} }
```

```
interface ObjectStack {
  void push(Object x);
  Object pop();
  Object peek();
  boolean empty();
}
```

objects of class Object reside on this stack; since Object is super class to any class, this stack allows any objects (even mixed!) as elements of it

assertions make sure that any preconditions on running the method successfully hold - they can get erased during runtime

- a stack allows elements to be put onto (**push**) and removed from (**pop**) the top of a list of elements
- an extra operation could be **peek**, that allows to get a reference to the top element without removal
- an operation **empty** could check if there are no elements on the stack
- the **interface ObjectStack** defines an **interface** for this behaviour for a list of **Objects**
- one can implement such a stack in many ways, e.g. using arrays as shown on the left

Downcasting

```
class ArrayObjectStack implements ObjectStack {
  Object[] stack;
  int size;
  int N;

  ArrayObjectStack() {
    N = 100;
    stack = new Object[N];
    size = 0;
  }

  public void push(Object x) {
    assert (size < 100);
    stack[size] = x;
    size = size + 1;
  }

  public Object pop() {
    assert (size > 0);
    Object result = stack[size-1];
    size = size - 1;
    return result;
  }

  public Object peek() {
    assert (size > 0);
    Object result =
      stack[size-1];
    return result;
  }

  public boolean empty() {
    return (size == 0);
} }
```

```
interface ObjectStack {
  void push(Object x);
  Object pop();
  Object peek();
  boolean empty();
}
```

- when retrieving objects from our **`ObjectStack`** we need to 'turn them' from type **`Object`** to the type we expect in order to access particular members – this is called 'casting'
- it is implemented using the **`(TargetClass)`** notation in front of the object to be cast

casting may create exceptions if we try to cast to a target that cannot be achieved

```
class StackWorld {

  public static void main (String[] args) {
    ObjectStack oStack = new ArrayObjectStack();
    oStack.push(new Robot("C3PO"));
    oStack.push(new TranslationRobot("a"));
    oStack.push(new CarrierRobot());
    //oStack.push(new Dog()); //will break the first cast!
    System.out.println(((Robot)oStack.pop()).name);
    System.out.println(((Robot)oStack.pop()).name);
    System.out.println(((Robot)oStack.peek()).name);
} }
```

A Dog is not a Robot!

cast of the object retrieved via peek to a Robot – the programmer is in charge to make sure the cast can be made