A1 a)

Make defensive copies when needed (Bloch-EJ Item 39)

• Make defensive copies of each mutable parameter to a constructor or to any mutator method (such as a set method)

–Perform defensive copying before checking validity

–Do not use clone to make a defensive copy of a parameter whose type can be sub-classed by untrusted parties

• For get methods, return defensive copies of mutable internal fields

A1 b)

A class is immutable if its instances cannot be modified

– The state of each instance of an immutable is fixed at instantiation –java.lang.String is an immutable class –java.util.Date is not immutable (but should be!)

advantages:

Immutables are simple and safe and make good building blocks

• An immutable is only ever in exactly one state

• Immutables are thread-safe

• Immutables can be cached

• There is no need to defensively copy or clone an immutable

• Mutable classes built on a foundation of immutable components are easier to maintain

• Safe as set elements or map keys (see Part 1.4)

• So: minimize mutability (Bloch-EJ Item 15)

A2 c)i)

Final class- one that you can not inhert from

Final fields- once initialized can not be changed

A2 c)ii)

Author has set methods

Book does not defensively copy

A2 c)iii)

public Book(Author author,String title,Date datePublished){

this.author = new Author(author.getFistName,author.getLastName);

this.title = title;

this.datePublished = new Date(datePublished.getTime());

}

public Author getAuthor(){

return new Author(author.getFirstName,author.getLastName);

}

public String getTitle(){

return title;

}

public Date getDatePublished(){

return (Date)datePublished.clone();

}

A2 a)

public static Account getInstance(String acc,int accNum)

A2 b)

Advantages compared to constructors:

1. Static factory methods can have more meaningful names

2. Static factory methods are not required to create a new object each time they are invoked

– Instances can be cached, can enforce other constraints

3. Static factory methods can return an object of any subtype of their return type

– The return type of an instantiated object can be restricted to its interface

– Implementation can be hidden and changed at run-time

– Service provider frameworks can be extended with new implementations (see Java Cryptography libraries)

A2 c)i)

because modulecode is final->there will only ever be one implementation

A2 c)ii)

1prefix should be non-null and not empty (length>0), can specify length=3

2throw null pointerexception if prefix is null. throw illegalArgument if prefix is empty or does not comply with specified length(can throw nullpointerexception as side effort of checking not empty)

3number should be>0(can also specify should be exactly 4 digits - in range of 1000to 9999 - for newcastle codes) throw illegalArgumentException if not in

A2 c)iii)

public final class ModuleCodes{

private final String prefix;

private final int number;

private static final Map<String,ModuleCode> CODES = HashMap<String,ModuleCode>();

  private ModuleCode(String prefix,int number){

  this.prefix = prefix;

  this.number = number;

  }

public static ModuleCode getInstance(String prefix,int number){

  final String code = prefix\*number;

  if(!CODES.containsKey(key)){

  CODES.put(code, new ModuleCode(prefix,number));

  return CODES.get(code);

  }

}

public String getPrefix(){

return prefix;

}

public int getNumber(){

return number;

}

}

A2 c)iV)

A3 a)

A1 a)

Make defensive copies when needed (Bloch-EJ Item 39)

• Make defensive copies of each mutable parameter to a constructor or to any mutator method (such as a set method)

–Perform defensive copying before checking validity

–Do not use clone to make a defensive copy of a parameter whose type can be sub-classed by untrusted parties

• For get methods, return defensive copies of mutable internal fields

A1 b)

A class is immutable if its instances cannot be modified

– The state of each instance of an immutable is fixed at instantiation –java.lang.String is an immutable class –java.util.Date is not immutable (but should be!)

advantages:

Immutables are simple and safe and make good building blocks

• An immutable is only ever in exactly one state

• Immutables are thread-safe

• Immutables can be cached

• There is no need to defensively copy or clone an immutable

• Mutable classes built on a foundation of immutable components are easier to maintain

• Safe as set elements or map keys (see Part 1.4)

• So: minimize mutability (Bloch-EJ Item 15)

A2 c)i)

Final class- one that you can not inhert from

Final fields- once initialized can not be changed

A2 c)ii)

Author has set methods

Book does not defensively copy

A2 c)iii)

public Book(Author author,String title,Date datePublished){

this.author = new Author(author.getFistName,author.getLastName);

this.title = title;

this.datePublished = new Date(datePublished.getTime());

}

public Author getAuthor(){

return new Author(author.getFirstName,author.getLastName);

}

public String getTitle(){

return title;

}

public Date getDatePublished(){

return (Date)datePublished.clone();

}

A2 a)

public static Account getInstance(String acc,int accNum)

A2 b)

Advantages compared to constructors:

1. Static factory methods can have more meaningful names

2. Static factory methods are not required to create a new object each time they are invoked

– Instances can be cached, can enforce other constraints

3. Static factory methods can return an object of any subtype of their return type

– The return type of an instantiated object can be restricted to its interface

– Implementation can be hidden and changed at run-time

– Service provider frameworks can be extended with new implementations (see Java Cryptography libraries)

A2 c)i)

because modulecode is final->there will only ever be one implementation

A2 c)ii)

1prefix should be non-null and not empty (length>0), can specify length=3

2throw null pointerexception if prefix is null. throw illegalArgument if prefix is empty or does not comply with specified length(can throw nullpointerexception as side effort of checking not empty)

3number should be>0(can also specify should be exactly 4 digits - in range of 1000to 9999 - for newcastle codes) throw illegalArgumentException if not in

A2 c)iii)

public final class ModuleCodes{

private final String prefix;

private final int number;

private static final Map<String,ModuleCode> CODES = HashMap<String,ModuleCode>();

  private ModuleCode(String prefix,int number){

  this.prefix = prefix;

  this.number = number;

  }

public static ModuleCode getInstance(String prefix,int number){

  final String code = prefix\*number;

  if(!CODES.containsKey(key)){

  CODES.put(code, new ModuleCode(prefix,number));

  return CODES.get(code);

  }

}

public String getPrefix(){

return prefix;

}

public int getNumber(){

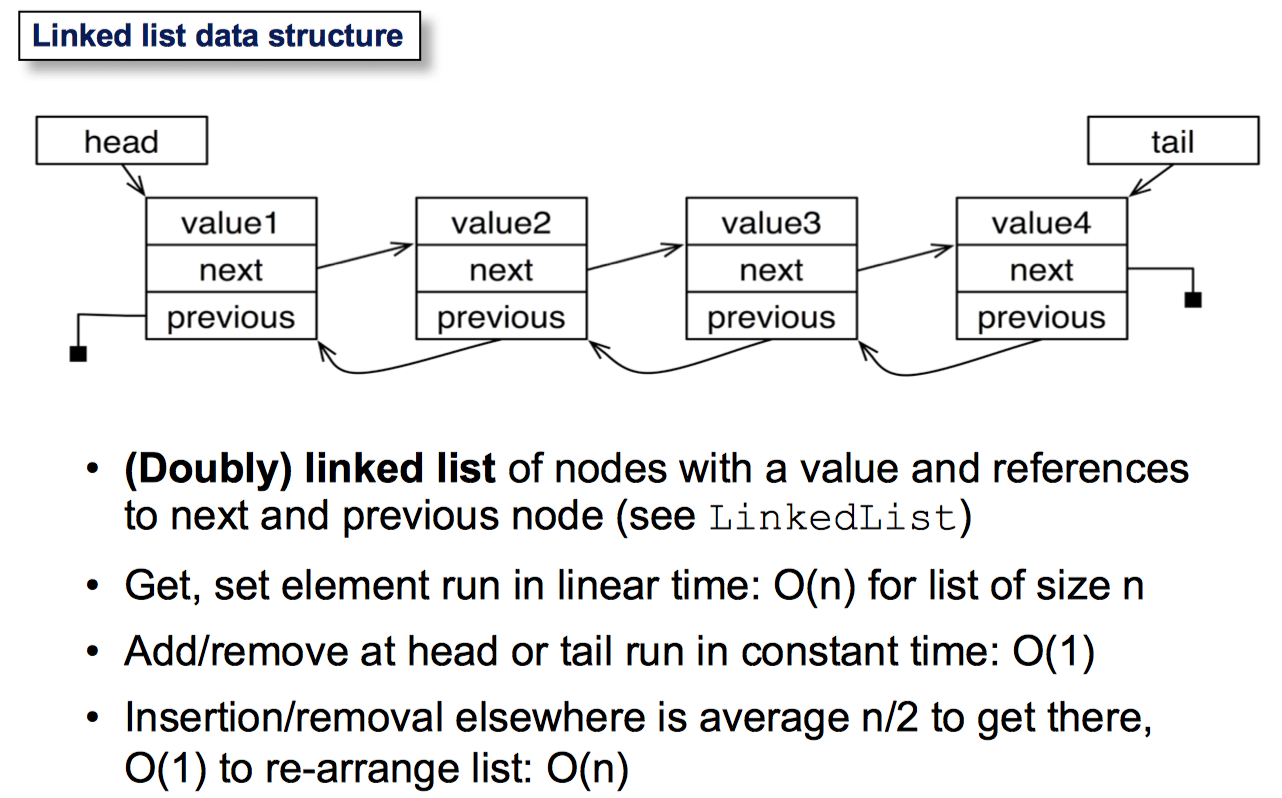
return number;

}

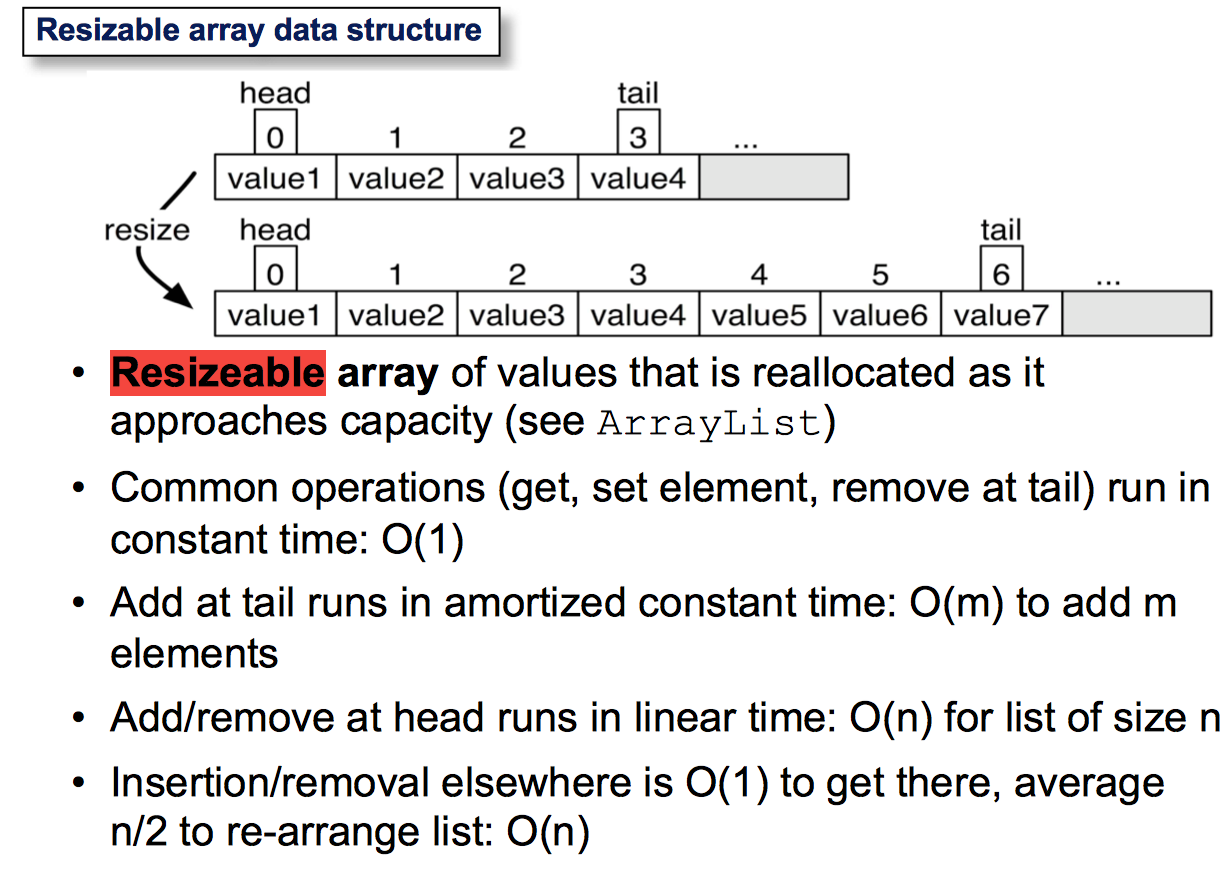
}

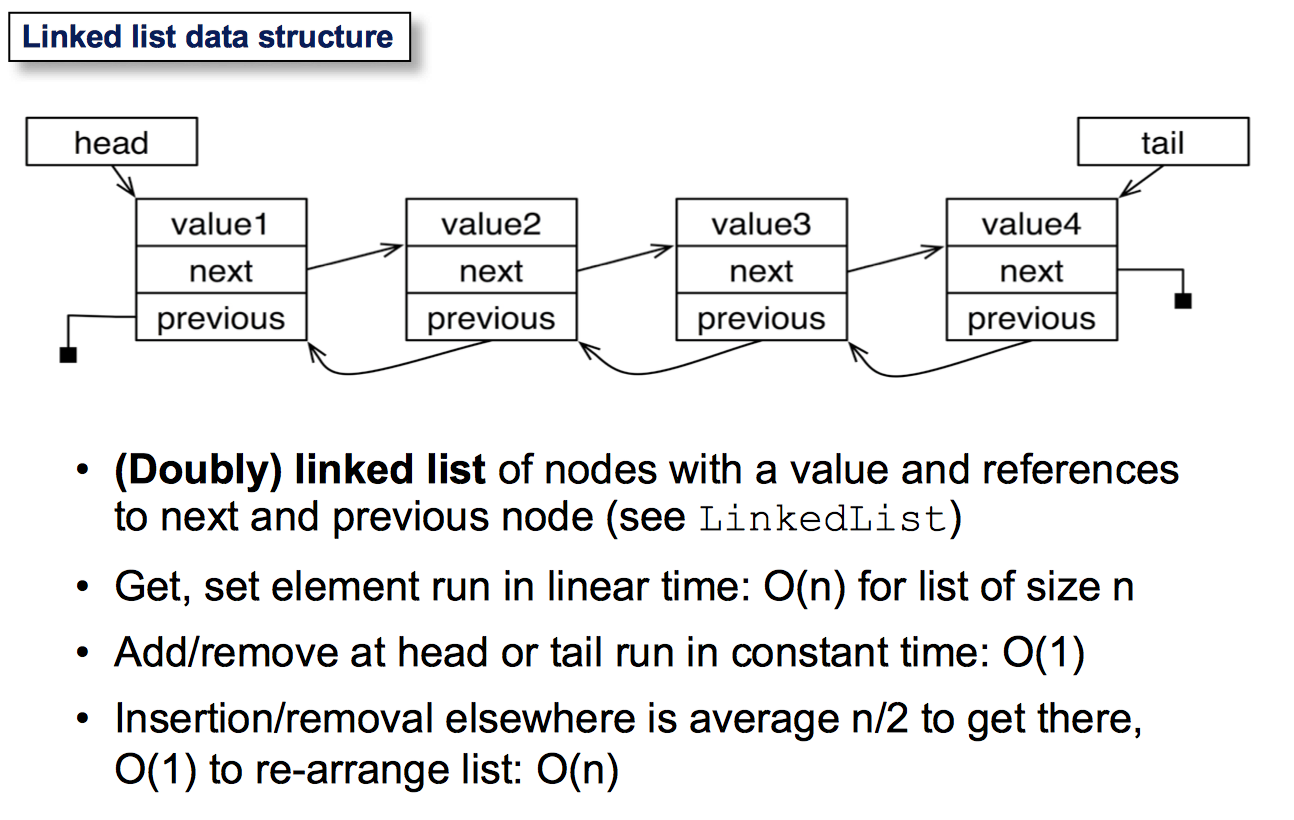
A2 c)iV)

A3 a)



A3 b)





A3 c)

NO

A3 d)